

FIG. 1

Table A: OPERATIONS	
ACQUIRE	
RELEASE	
CONNECT	
DISCONNECT	
GET INFO	
GET CONTEXT	
SET CONTEXT	
OPEN QUERY	
GET FIRST	
GET NEXT	
GET PREV	
GET LAST	
CLOSE QUERY	

Table B: CONTROL BLOCK (Used with all Operations)	
NAME	STRING
ID	NUMBER
CONNECTION_ID	NUMBER
TYPE	ENUM: FUNCTION, MESSAGE, INTERFACE...
DATA	BINARY WITH SIZE
CONTEXT	NUMBER

Table C: COMMON TERMINAL INFORMATION FIELDS	
NEGOTIATION	ENUM: FIXED, NEGOTIABLE
DIRECTION	ENUM: IN, OUT, BI-DIR, OTHER
SYNCHRONOSITY	ENUM: SYNC, AMBIENT, ASYNC
CARDINALITY	NUMBER
CONTRACT NAME	STRING
CONTRACT ID	BINARY WITH SIZE

(Other Fields can be defined)
All fields on the terminal info descriptor are returned together in the data field of the control block by GET INFO operation



FIG. 2

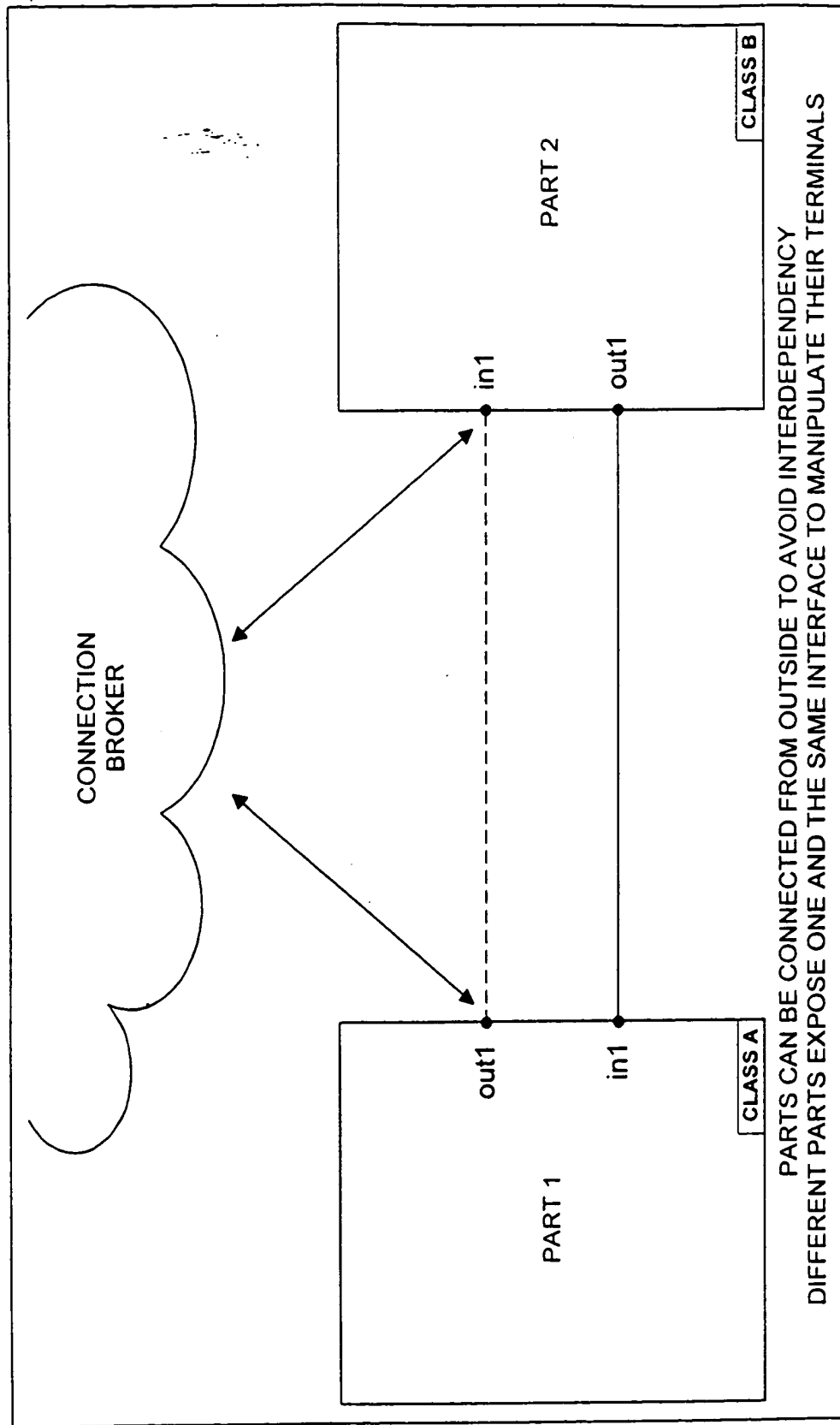


FIG. 3

Table A: TERMINAL NAMES ARE:
<ul style="list-style-type: none"> • UNIQUE WITHIN THE PART • CAN BE CONSTRUCTED HIERARCHICALLY • INTERPRETED BY THE PART • CAN REPRESENT COMPLEX STRUCTURES

Table B: EXAMPLES OF VALID NAMES
ARM_CONTROL SENSOR [1].CONTROL SENSOR [1].OUTPUT SENSOR [2].OUTPUT MOTOR.MAIN MOTOR.AVX OUTPUT [3]

Table D: SIMPLE SET OF TERMINALS
CONTROL STATUS DATA NOTIFICATIONS_IN NOTIFICATIONS_OUT

Table C: EXAMPLES OF INVALID NAMES
[4] TEST] SENSOR.] MEMORY.ALLOC MEMORY [ALLOC]

Table E: ARRAY OF TERMINALS
ARM_CONTROL [0] ARM_CONTROL [1] ARM_CONTROL [2]

Table F: STRUCTURE OF TERMINALS
ARM_CONTROL.STATUS ARM_CONTROL.POSITION ARM_CONTROL.GRIP

FIG. 4

TYPE	DATA PROVIDED IN THE DATA FIELD OF CONTROL BLOCK
FUNCTION	POINTER TO FUNCTION, CONTEXT, CONTRACT ID
MESSAGE	OBJECT ID, MESSAGE ID, CONTRACT ID
MESSAGE INTERFACE	OBJECT ID, CONTRACT ID
VTBL INTERFACE	POINTER TO INTERFACE, CONTRACT ID
OLE INTERFACE	POINTER TO INTERFACE, OLE ID
VTBL PLUG	POINTER TO INTERFACE, CONTRACT ID
PIPE	FILE HANDLE, CONTRACT ID

THE DATA ABOVE IS EXCHANGED BETWEEN TERMINALS IN THE PROCESS OF ESTABLISHING OF CONNECTION USING OPERATIONS ACQUIRE AND CONNECT OF THE TERMINAL INTERFACE

FIG. 5

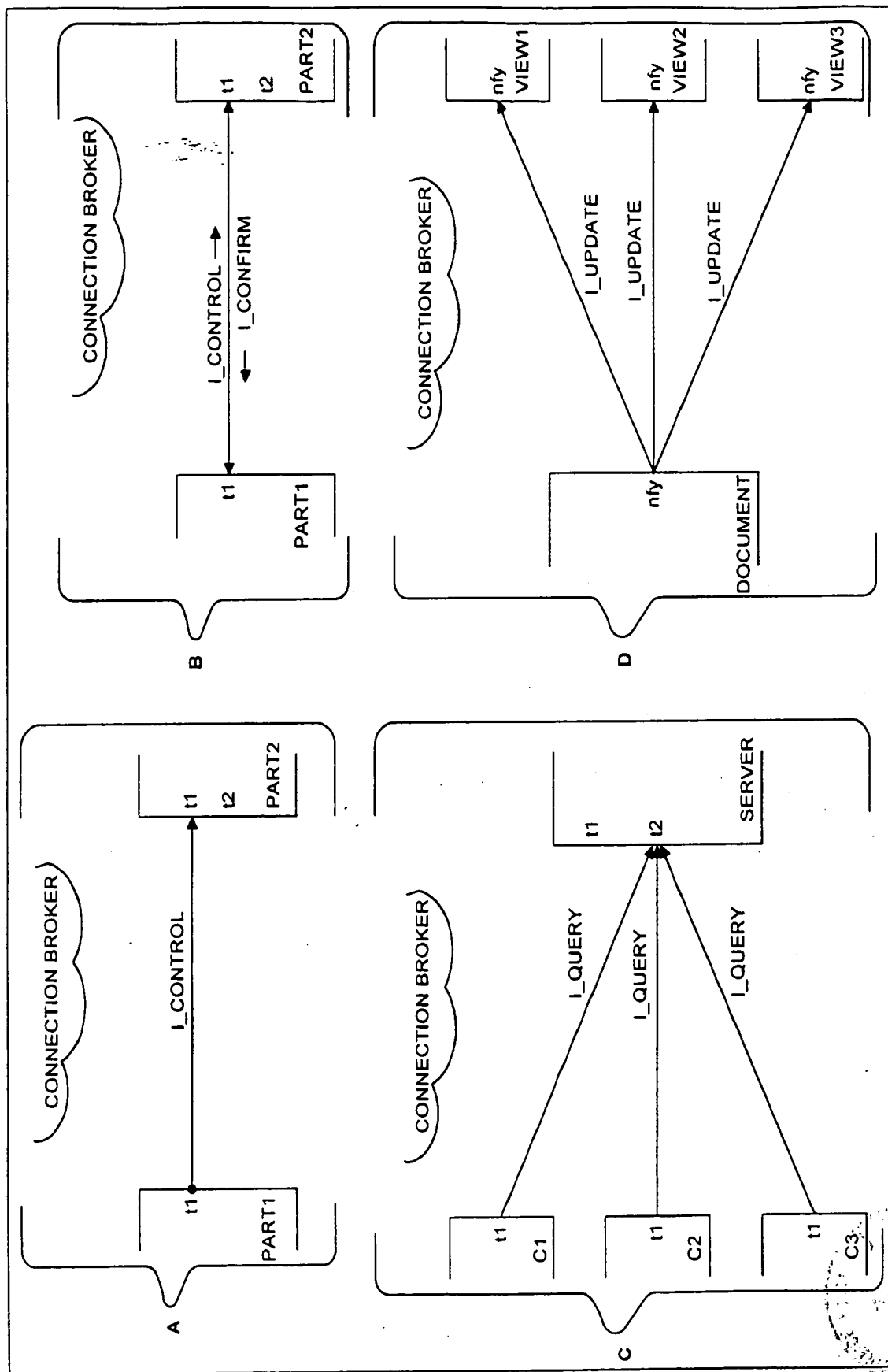


FIG. 6

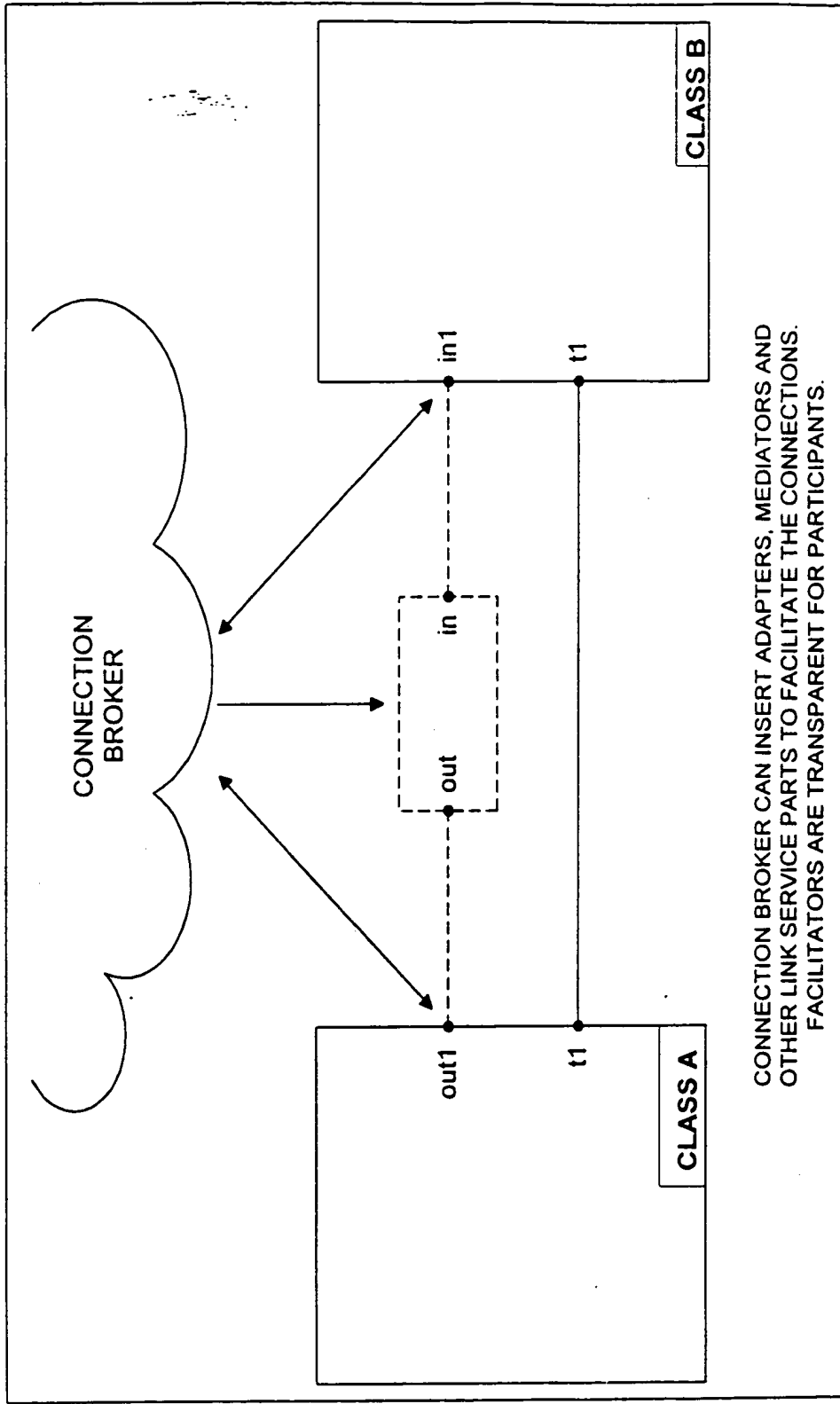


FIG. 7

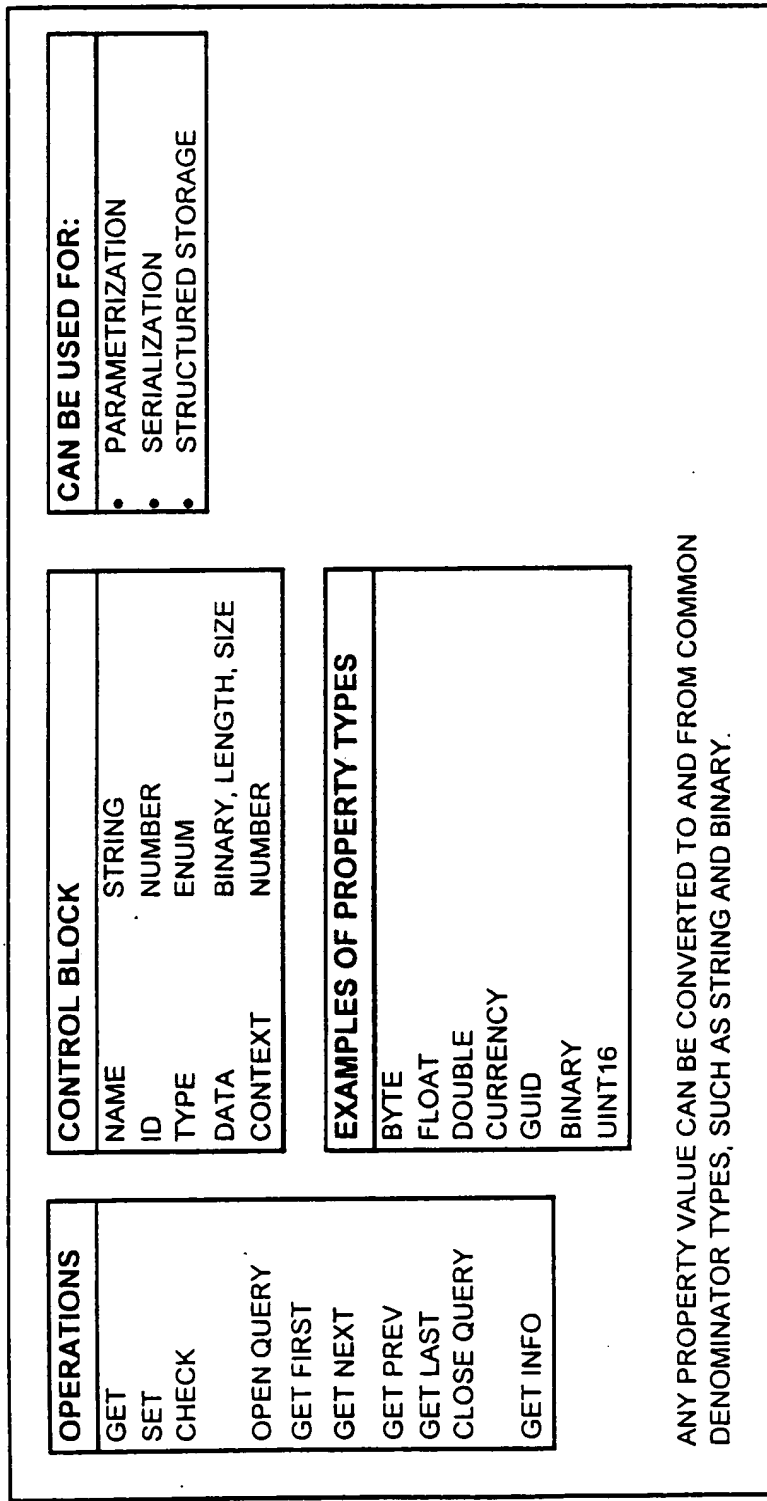


FIG. 8

Table A: PROPERTY NAMES ARE:

- UNIQUE WITH THE PART
- CAN BE CONSTRUCTED HIERARCHICALLY
- INTERPRETED BY THE PART
- CAN REPRESENT COMPLEX DATA STRUCTURES
- QUERIES CAN BE OPEN BY PARTIAL NAMES USING WILDCARDS
- OTHER OPERATIONS ON PARTIAL NAMES MEAN OPERATION IS PERFORMED ON A SET OF PROPERTIES THE NAMES OF WHICH SATISFY THE WILDCARD TEMPLATE

Table C: EXAMPLES: PARTIAL NAMES

- A. TRAIN [*]. CONDUCTOR.NAME
- B. TRAIN [3]. CONDUCTOR.*
- C. TRAIN [3].*.NAME

Table B: EXAMPLES: VALID AND INVALID NAMES

VALID:	WEIGHT DATA [1] SENSOR.PRESSURE
INVALID:	READOUT] SENSOR.]

Table D: EXAMPLES: SIMPLE NAMES

BUFFER_SIZE
MAX_BUFFERS
PRIORITY
STORAGE_LOCATION

Table E: EXAMPLES: ARRAYS AND STRUCTURES

READOUT [0]	SENSOR.PRESSURE
READOUT [1]	SENSOR.TEMPERATURE
READOUT [2]	SENSOR.STATE

FIG. 9

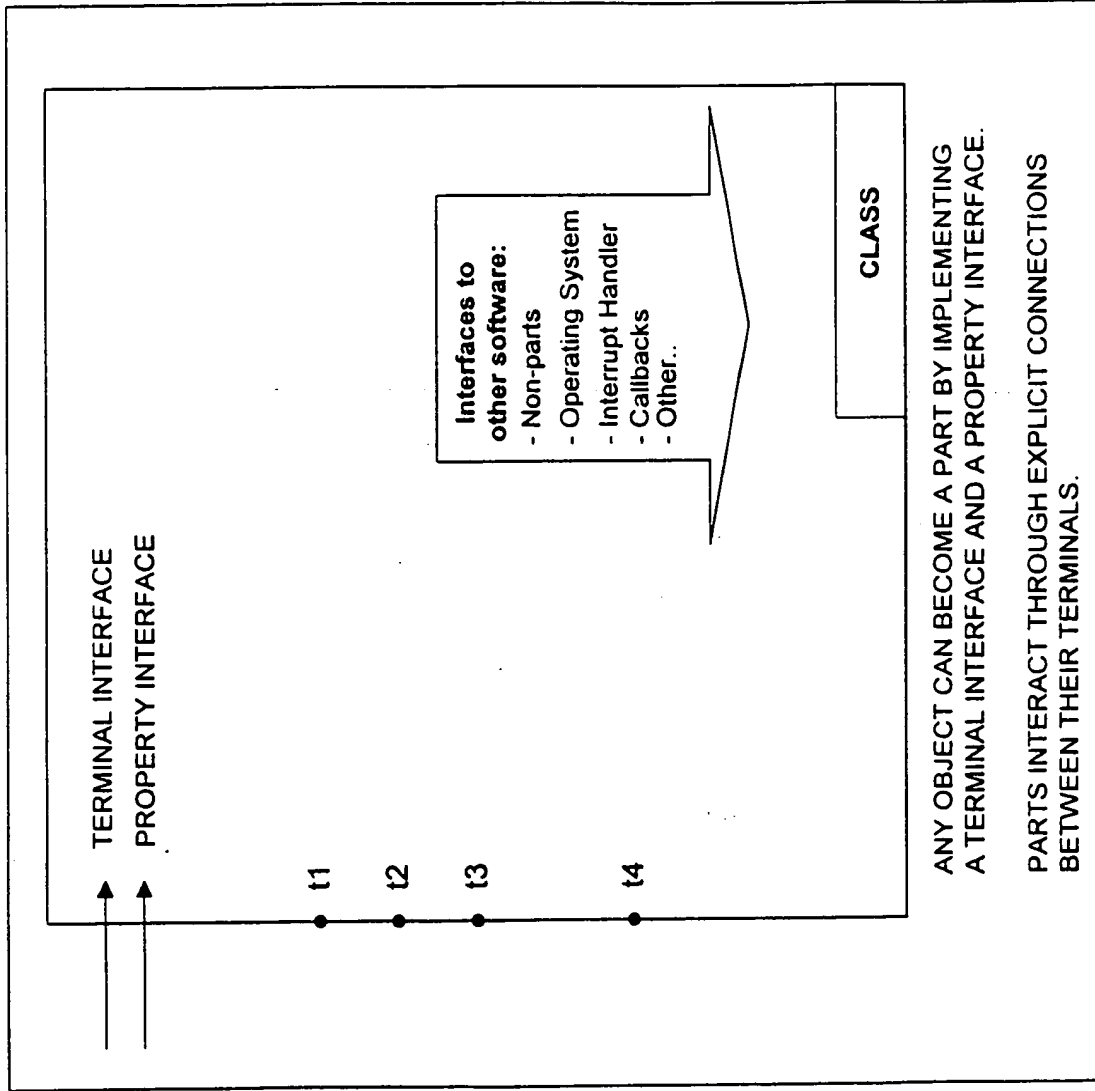


FIG. 10

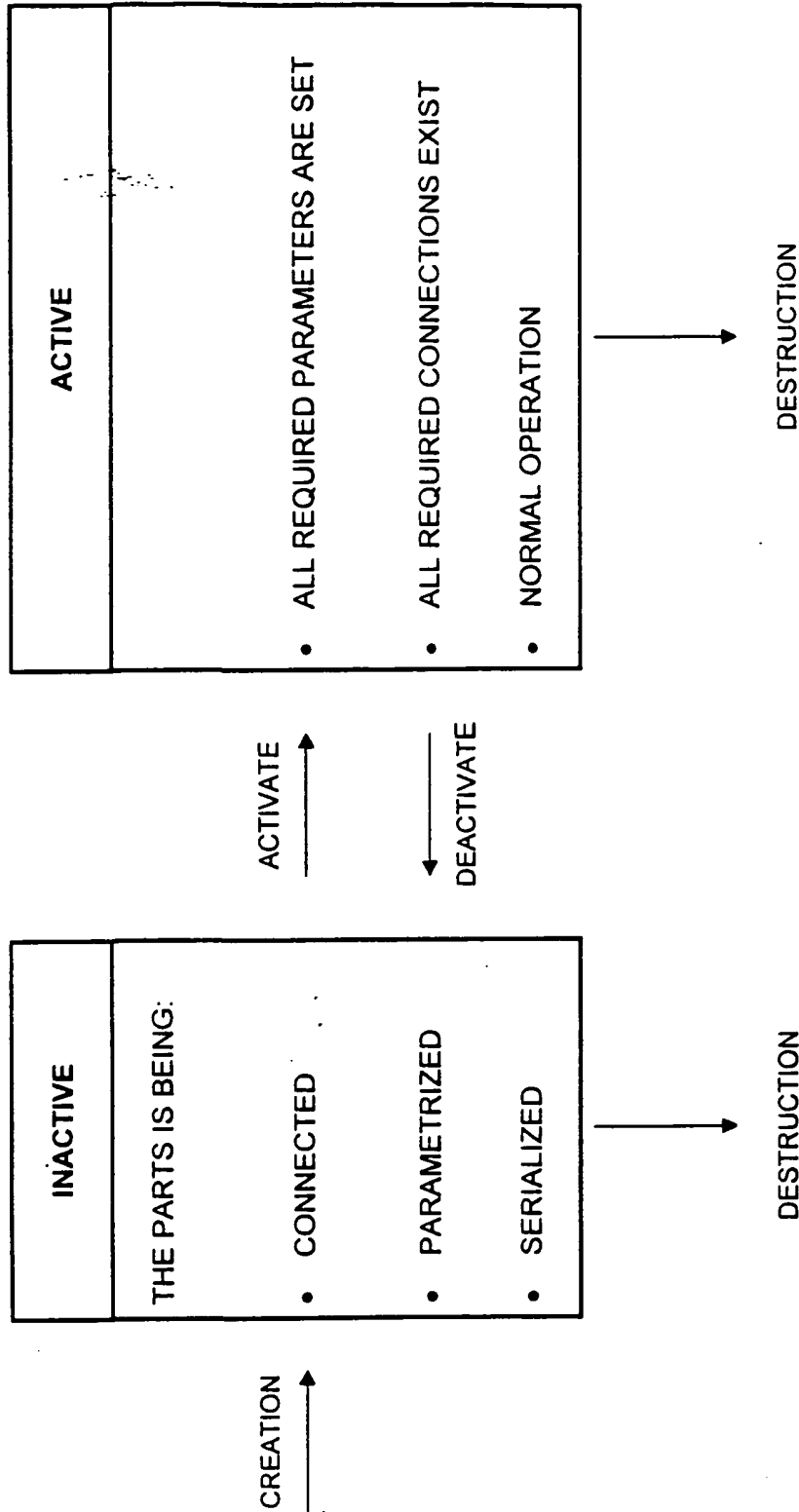


FIG. 11



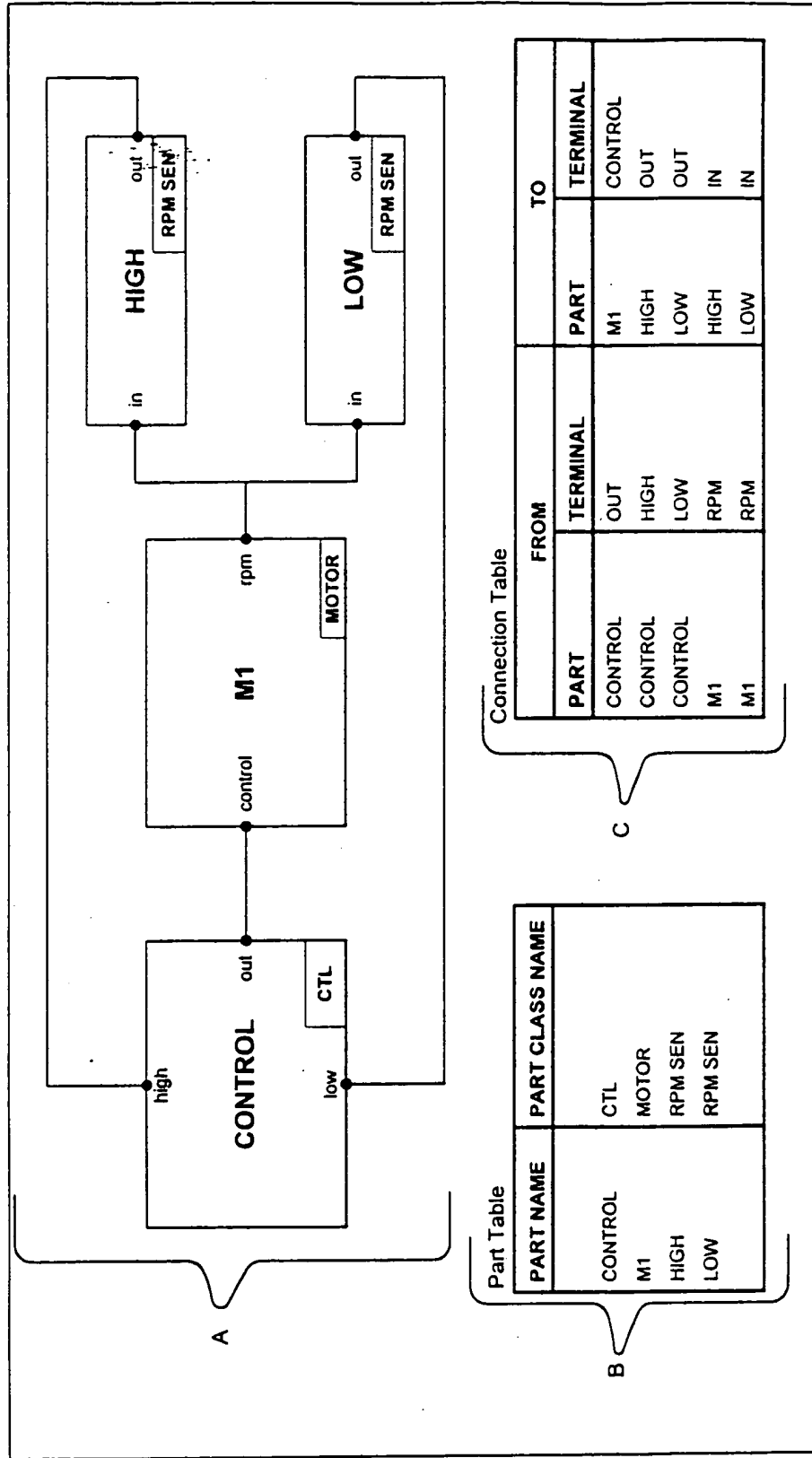


FIG. 12

Table A:

CONTROL	HIGH	CONTROL.ENABLED
	THRESHOLD	M1.STATE
ENABLED	IOADDR	M1.IOADDR
	IRQ	M1.IRQ
	SENSITIVITY	HIGH.THRESHOLD
		HIGH.IOADDR
M1		HIGH.IRQ
		LOW.THRESHOLD
STATE	LOW	LOW.IOADDR
	THRESHOLD	LOW.IRQ
IOADDRESS	IOADDR	HIGH.SENSITIVITY
	IRQ	LOW.SENSITIVITY
	SENSITIVITY	



Table B: HARD PARAMETRIZATION TABLE

PATH	VALUE
M1.IOADDRESS	0x300
M1.IRQ	-1
HIGH.IOADDR	0x100
HIGH.IRQ	10
LOW.IOADDR	0x120
LOW.IRQ	11

Table D: GROUP PROPERTY

SENSITIVITY	HIGH.SENSITIVITY
	LOW.SENSITIVITY

Table C: ALIAS TABLE

BASE PATH	G	S	TARGET PATH
MIN_RPM	Y	Y	LOW.THRESHOLD
MAX_RPM	Y	Y	HIGH.THRESHOLD
STATE	Y	N	M1.STATE
ENABLED	Y	Y	CONTROL.ENABLED
HIGH.IOADDR	N	N	N/A
HIGH.IRQ	N	N	N/A
LOW.IOADDR	N	N	N/A
LOW.IRQ	N	N	N/A
M1.IOADDRESS	N	N	N/A
M1.IRQ	N	N	N/A

FIG. 13

FD-302 (Rev. 11-29-60)

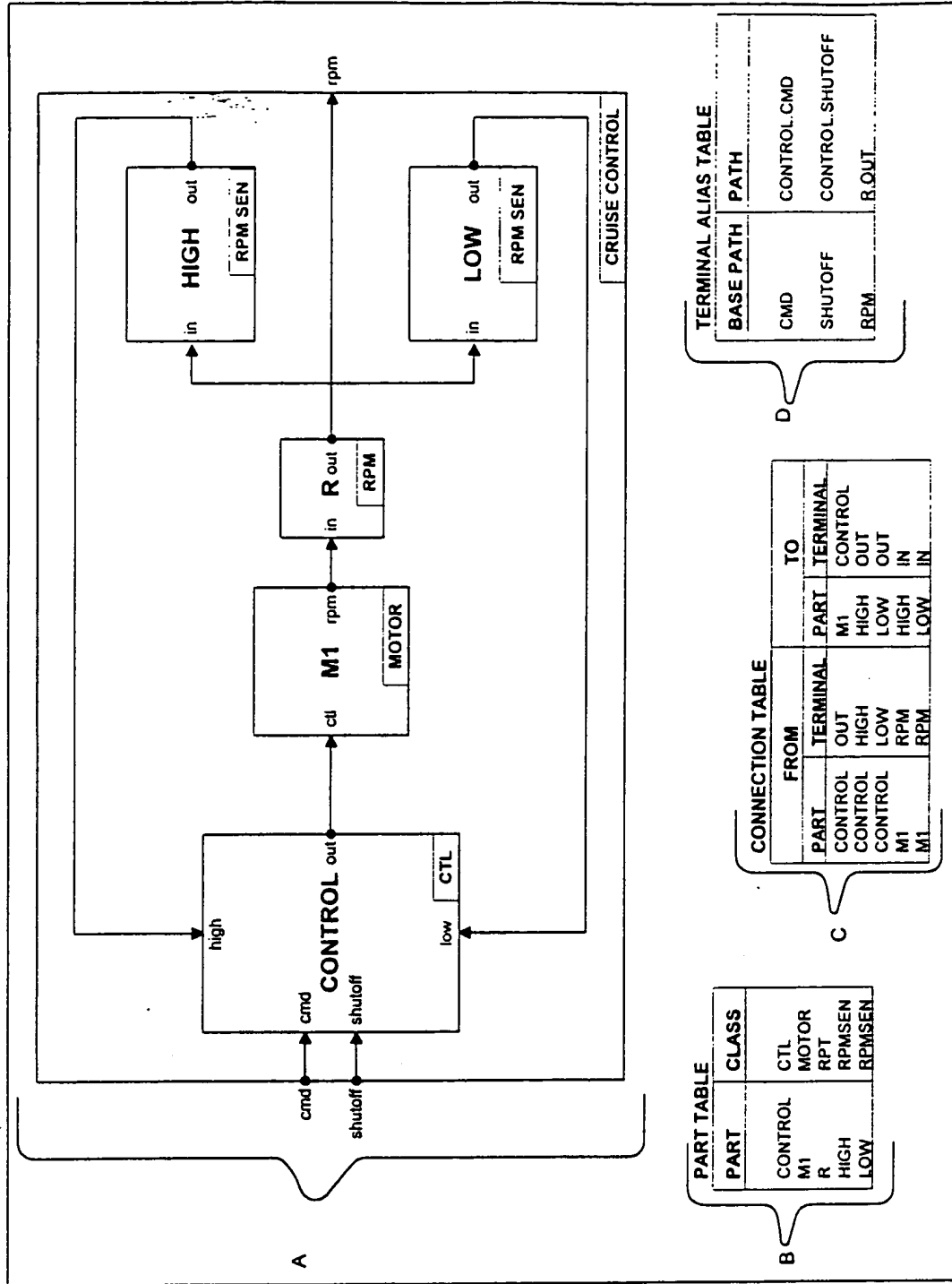


FIG. 14

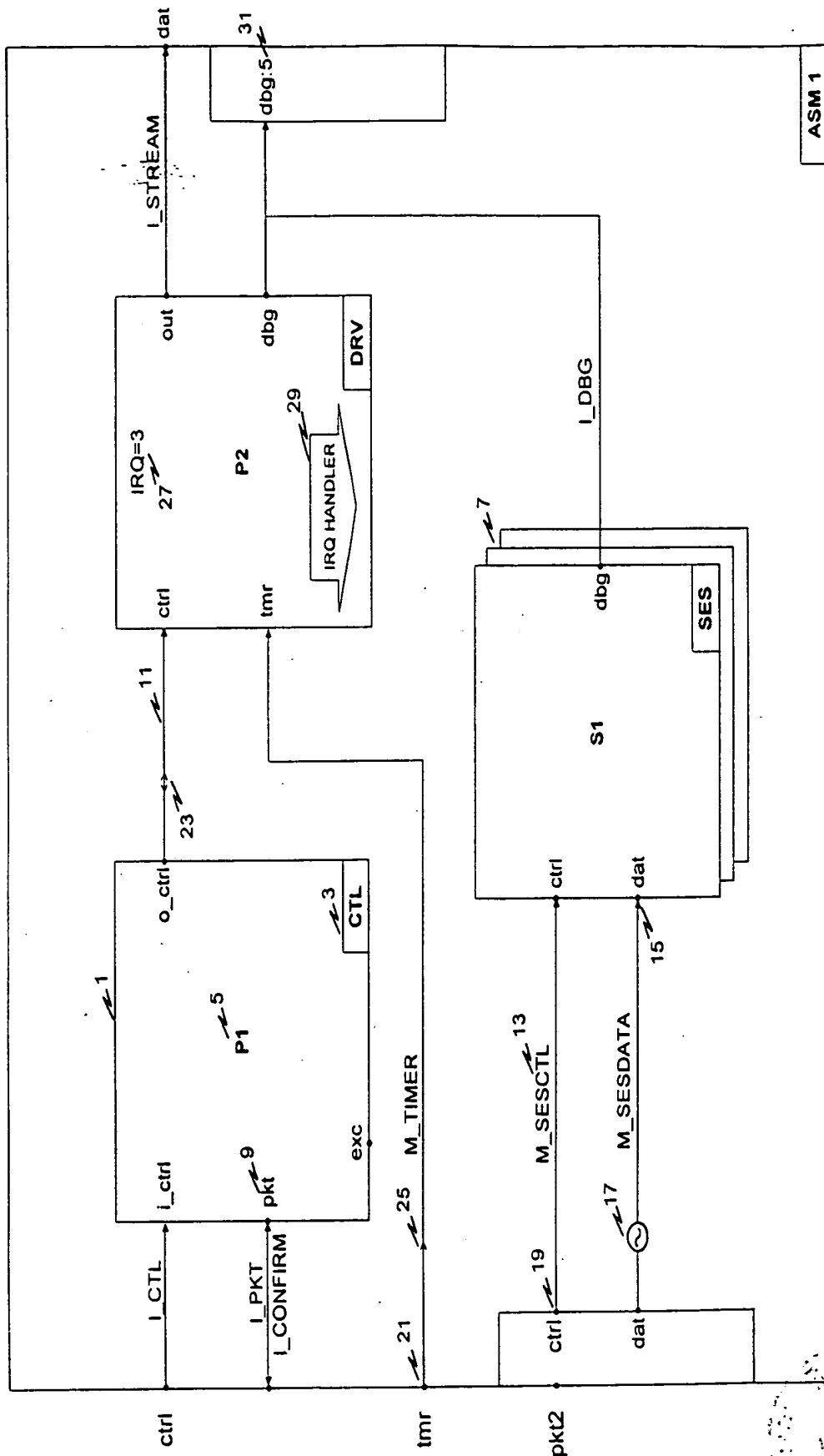


FIG. 15

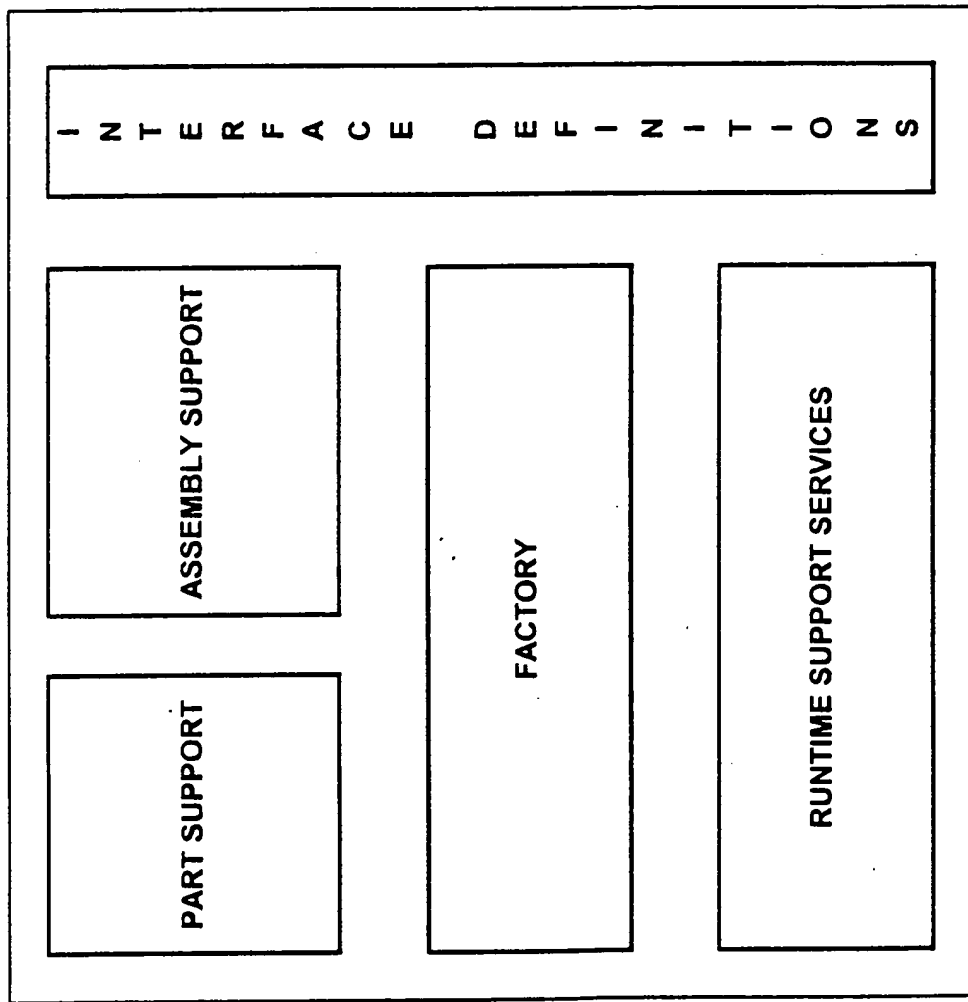


FIG. 16


```

1  /* ----- */
2  /*          ETC: Global system definitions          */
3  /*          */
4  /*          TYPES.H - Portable type definitions      */
5  /* ----- */
6
7  #ifndef _TYPES_DEFINED
8  #define _TYPES_DEFINED
9
10 #if (!defined(BIT32) && !defined(BIT16))
11     #error BIT16 or BIT32 must be defined
12 #endif
13
14 /* --- Common types ----- */
15
16 typedef int  bool;
17
18 #ifndef TRUE
19     #define FALSE    ((bool) 0)
20     #define TRUE     ((bool) (!FALSE))
21 #endif
22
23 typedef signed char  schar;
24 typedef unsigned char  uchar;
25
26 #ifndef _WINNT_
27     #define MINCHAR    ((char) 0x00)        // 0x80 in WINNT.H!!
28     #define MAXCHAR    ((char) 0x7F)
29 #endif
30
31 #define MINSCHAR ((char) 0x80)
32 #define MAXSCHAR ((char) 0x7F)
33
34 #define MINUCHAR ((char) 0x00)
35 #define MAXUCHAR ((char) 0xFF)
36
37 typedef void  *_datap;
38 typedef void  *_codep; // $$ "void *_codep (void);" would not work
39
40 #ifdef __cplusplus
41     #define NULLD    0
42     #define NULLC    0
43 #else
44     #define NULLD (( _datap) 0)
45     #define NULLC (( _codep) 0)
46 #endif
47
48 #ifndef NULL
49     #define NULL ((void *) 0)
50 #endif
51
52 /*
53     Notes:
54
55     1. Types _datap and _codep are provided for systems/models in which
56        data pointers are of different size than code pointers. See also
57        type _pointer in the compatibility types section of this header file.
58 */
59

```

FIG. 17A

```

1  /* --- Machine types ----- */
2
3  typedef unsigned      char byte;
4  typedef unsigned short int word;
5  typedef unsigned long int dword;
6
7  #define MINBYTE ((byte) 0x00)
8  #ifndef _WINNT_
9      #define MAXBYTE ((byte) 0xFF)
10 #endif
11
12 #define MINWORD ((word) 0x0000U)
13 #ifndef _WINNT_
14     #define MAXWORD ((word) 0xFFFFU)
15 #endif
16
17 #define MINDWORD ((dword) 0x00000000UL)
18 #ifndef _WINNT_
19     #define MAXDWORD ((dword) 0xFFFFFFFFUL)
20 #endif
21
22 /* --- Tolerance types (hardware-dependent) ----- */
23
24 typedef unsigned int uint;    //guaranteed to be at least 16 bits wide.
25 typedef signed int sint;     //guaranteed to be at least 16 bits wide.
26
27 #define MININT ((int) 0x0000)
28 #define MAXINT ((int) 0x7FFF)
29
30 #define MINSINT ((sint) -32767-1)
31 #define MAXSINT ((sint) 0x7FFF)
32
33 #define MINUINT ((uint) 0x0000U)
34 #define MAXUINT ((uint) 0xFFFFU)
35
36 /*
37  Notes:
38
39  1. Use tolerance types for variables that are expected to only take
40     small values which will never exceed the minimum number of guaranteed
41     bit width, so that the compiler can choose the actual type which the
42     underlying hardware feels most comfortable in dealing with.
43
44  2. Use of tolerance types is only allowed for variables of a local scope;
45     it is prohibited for public/external variables. Writing the binary
46     contents of a tolerance variable to a disk file is punishable by death.
47
48  3. Negative values are not allowed for type int. (thus it is 15 bits wide
49     instead of 16 bits wide.) Use type sint if you are planning to assign
50     negative values to a variable.
51
52  4. Alternatively, type int can be prohibited by #defining it as "$$$"
53     (in the end of this header file) and the use of uint or sint can be
54     promoted instead.
55
56  5. Operations that result in overflow/underflow beyond the minimum ..
57     number of guaranteed bits yield undefined results and should be
58     avoided for all tolerance types.
59  */

```

FIG. 17B

104030 26920360

```

1
2  /* --- Portable types ----- */
3
4  typedef          char  int8 ;
5  typedef signed   char  sint8 ;
6  typedef unsigned char  uint8 ;
7  typedef          short int  int16;
8  typedef signed   short int  sint16;
9  typedef unsigned short int  uint16;
10 typedef          long  int  int32;
11 typedef signed   long  int  sint32;
12 typedef unsigned long  int  uint32;
13
14 typedef unsigned          char  flg8 ;
15 typedef unsigned short int  flg16;
16 typedef unsigned long  int  flg32;
17
18 typedef uint32             _ctx;           // max. size between _datap & uint32
19 #define NO_CTX (( _ctx)0)
20
21
22 typedef uint32             _id32;         // 32-bit id
23 #define NO_ID32 (( _id32)0)
24
25 typedef uint16             _id16;         // 16-bit id
26 #define NO_ID16 (( _id16)0)
27
28 typedef _id32              _id;           // default id (32-bit)
29 #define NO_ID (( _id)0)
30
31
32 typedef int8               bool8;         // the shortest stand-alone boolean
33
34 #define MININT8 (( int8) 0x00)
35 #define MAXINT8 (( int8) 0x7F)
36
37 #define MINSINT8 ((sint8) 0x80)
38 #define MAXSINT8 ((sint8) 0x7F)
39
40 #define MINUINT8 ((uint8) 0x00)
41 #define MAXUINT8 ((uint8) 0xFF)
42
43 #define MININT16 (( int16) 0x0000)
44 #define MAXINT16 (( int16) 0x7FFF)
45
46 #define MINSINT16 ((sint16) 0x8000)
47 #define MAXSINT16 ((sint16) 0x7FFF)
48
49 #define MINUINT16 ((uint16) 0x0000U)
50 #define MAXUINT16 ((uint16) 0xFFFFU)
51
52 #define MININT32 (( int32) 0x00000000L)
53 #define MAXINT32 (( int32) 0x7FFFFFFFL)
54
55 #define MINSINT32 ((sint32) 0x80000000L)
56 #define MAXSINT32 ((sint32) 0x7FFFFFFFL)
57
58 #define MINUINT32 ((uint32) 0x00000000UL)
59 #define MAXUINT32 ((uint32) 0xFFFFFFFFUL)

```

FIG. 17C

```

1
2  /*
3  Notes:
4
5  1.  Use int16 and int32 when you know the exact bit width that you want
6      for your variable, but you do not care about the interpretation of
7      the sign of the variable.  This way, the compiler will choose an
8      actual signed or unsigned type, depending on what the underlying
9      hardware feels most comfortable in dealing with.
10
11  2.  As an example, variables that are ideal candidates to be declared as
12      int16 or int32 are sets of bitflags: although the bit capacity of such
13      a variable is crucial, the variable will never be interpreted as a
14      signed or unsigned value.
15
16  3.  All integer operations defined by the C language are valid on all of
17      the portable types, and overflow/underflow operations beyond
18      the specified bit widths have predictable behaviour.
19  */
20
21 #endif // defined _TYPES_DEFINED
22

```

FIG. 17D

```

1  /* ----- */
2  /*          ETC: Global system definitions          */
3  /*          */
4  /*          STATUS.H - global system statae        */
5  /* ----- */
6
7  #ifndef _STATUS_DEFINED
8  #define _STATUS_DEFINED
9
10 /* --- define status type ----- */
11
12 #define _stat uint
13
14 /* --- define if_xxx macros ----- */
15
16 #define if_ret(s,e)      if (((s) = (e)) != ST_OK) return ((s))
17 #define if_cleanup(s,e,c) if (((s) = (e)) != ST_OK) return (((c), (s)))
18
19 /* --- define general status values ----- */
20
21 #define ST_OK            0 // successful operation
22
23 /* structures/tables manipulation */
24 #define ST_ALLOC         20 // can't alloc (for variable size alloc/pool)
25 #define ST_NO_ROOM       21 // can't alloc fixed sz entry in fixed sz pool
26 #define ST_OVERFLOW      22 // value, buffer, whatever may overflow
27 #define ST_UNDERFLOW     23 // value, buffer, counter, whatever may underflow
28 #define ST_EMPTY         24 // QUEUE/smith is empty - no elements
29 #define ST_FULL          25 // queue/smith is full
30 #define ST_EOF           26 // end of file
31
32 /* valchk */
33 #define ST_INVALID       30 // value not valid in operation context
34 #define ST_BAD_VALUE     31 // value not valid in module's context
35 #define ST_OUT_OF_RANGE  32 // value out of range
36 #define ST_NULL_PTR      33 // NULL ptr is not valid in operation context
37 #define ST_BAD_SYNTAX    34 // text/formatted data syntax error
38 #define ST_BAD_NAME      35 // name invalid in module's context
39
40 /* BAD errors */
41 #define ST_UNEXPECTED     40 // unexpected condition of parm or external object
42 #define ST_PANIC          41 // unexpected condition of self or internal object
43 #define ST_DEADLOCK       42 // deadlock detected
44
45 /* failures */
46 #define ST_REFUSE         45 // oper rejected voluntarily (inappropriate state)
47 #define ST_NO_ACTION      46 // oper requires no action (eg close closed file)
48 #define ST_FAILED        47 // operation failed / not successful
49
50 /* state check */
51 #define ST_NOT_INITED     50 // object is not initialized
52 #define ST_NOT_ACTIVE     51 // oper allowed only when object is active
53 #define ST_NOT_OPEN       52 // oper allowed only when object is open
54
55 /* i/o system */
56 #define ST_IOERR          60 // I/O error - file system / peripherals error
57 #define ST_BAD_CHKSUM     61 // wrong chksum
58
59 /* set-related */

```

FIG. 18A

```

1  #define ST_NOT_FOUND          62 // requested item not found
2  #define ST_DUPLICATE         63 // duplicated item is not allowed
3
4  /* critical section / privileges */
5  #define ST_BUSY               70 // in critical section (dynamic, short period)
6  #define ST_ACCESS_DENIED     71 // temporary lack of access (dynamic, long period)
7  #define ST_PRIVILEGE         72 // privilege violation (static)
8  #define ST_SCOPE_VIOLATION   73 // (no comment)
9  #define ST_BAD_ACCESS        74 // access type not appropriate / not possible
10
11 /* operation status */
12 #define ST_PENDING            80 // operation is pending
13 #define ST_TIMEOUT            81 // timeout has expired
14 #define ST_CANCELLED          82 // operation cancelled on user's request
15 #define ST_ABORTED            83 // op. aborted on system/module's request
16 #define ST_RESET              84 // object has been orderly reset
17 #define ST_CLEANUP             85 // object is about to be destroyed, now in cleanup
18 #define ST_OVERRIDE           86 // operation/entity was overridden
19
20 /* API */
21 #define ST_CANT_BIND           90 // can't bind / resolve name, handle, etc.
22 #define ST_FPI_ERROR           91 // invalid function request, etc.
23 #define ST_WRONG_VERSION       92 // incompatible version of data/code detected
24 #define ST_NOT_IMPLEMENTED     93 // feature not implemented
25 #define ST_NOT_SUPPORTED       94 // partial implementation; feature not supported
26
27 /* environment-specific */
28 #define ST_BAD_UID             100 // OBJ:invalid uid
29 #define ST_BAD_MSG             101 // OBJ:invalid msg or object doesn't handle it
30 #define ST_OK_CUT              102 // OBJ:don't call more virt. objs; return ST_OK
31 #define ST_NOT_INHERITED       103 // OBJ:method was abstract; needs to be inherited
32 #define ST_BAD_PID            104 // OBJ:invalid pid
33 #define ST_WIN_DFLT            210 // WIN:WindowProc: call DefWinProc, use its result
34 #define ST_BLOCK_OVERRIDE     300 // KER:attempt to block a thread that is already
35 blocked
36
37 /* --- String table ----- */
38
39 #ifdef _ERR_STAT
40
41 struct STAT_TXT_S
42 {
43     _stat s;
44     char *p;
45 };
46
47 #define DEF_STATUS(s)          ( (s), #s )
48
49 static struct STAT_TXT_S _err_sts_tbl[] =
50 {
51     DEF_STATUS ( ST_OK ),
52     DEF_STATUS ( ST_ALLOC ),
53     DEF_STATUS ( ST_NO_ROOM ),
54     DEF_STATUS ( ST_OVERFLOW ),
55     DEF_STATUS ( ST_UNDERFLOW ),
56     DEF_STATUS ( ST_EMPTY ),
57     DEF_STATUS ( ST_FULL ),
58     DEF_STATUS ( ST_EOF ),
59     DEF_STATUS ( ST_INVALID ),

```

FIG. 18B

```

1      DEF_STATUS ( ST_BAD_VALUE          ),
2      DEF_STATUS ( ST_OUT_OF_RANGE       ),
3      DEF_STATUS ( ST_NULL_PTR           ),
4      DEF_STATUS ( ST_BAD_SYNTAX         ),
5      DEF_STATUS ( ST_BAD_NAME           ),
6      DEF_STATUS ( ST_UNEXPECTED         ),
7      DEF_STATUS ( ST_PANIC              ),
8      DEF_STATUS ( ST_DEADLOCK           ),
9      DEF_STATUS ( ST_REFUSE              ),
10     DEF_STATUS ( ST_NO_ACTION           ),
11     DEF_STATUS ( ST_FAILED              ),
12     DEF_STATUS ( ST_NOT_INITED          ),
13     DEF_STATUS ( ST_NOT_ACTIVE          ),
14     DEF_STATUS ( ST_NOT_OPEN            ),
15     DEF_STATUS ( ST_IOERR               ),
16     DEF_STATUS ( ST_BAD_CHKSUM          ),
17     DEF_STATUS ( ST_NOT_FOUND           ),
18     DEF_STATUS ( ST_DUPLICATE           ),
19     DEF_STATUS ( ST_BUSY                ),
20     DEF_STATUS ( ST_ACCESS_DENIED       ),
21     DEF_STATUS ( ST_PRIVILEGE           ),
22     DEF_STATUS ( ST_SCOPE_VIOLATION     ),
23     DEF_STATUS ( ST_BAD_ACCESS          ),
24     DEF_STATUS ( ST_PENDING             ),
25     DEF_STATUS ( ST_TIMEOUT             ),
26     DEF_STATUS ( ST_CANCELLED           ),
27     DEF_STATUS ( ST_ABORTED             ),
28     DEF_STATUS ( ST_RESET               ),
29     DEF_STATUS ( ST_CLEANUP             ),
30     DEF_STATUS ( ST_OVERRIDE            ),
31     DEF_STATUS ( ST_CANT_BIND           ),
32     DEF_STATUS ( ST_FPI_ERROR           ),
33     DEF_STATUS ( ST_WRONG_VERSION       ),
34     DEF_STATUS ( ST_NOT_IMPLEMENTED     ),
35     DEF_STATUS ( ST_NOT_SUPPORTED       ),
36     DEF_STATUS ( ST_BAD_UID             ),
37     DEF_STATUS ( ST_BAD_MSG             ),
38     DEF_STATUS ( ST_OK_CUT              ),
39     DEF_STATUS ( ST_NOT_INHERITED       ),
40     DEF_STATUS ( ST_BAD_PID             ),
41     DEF_STATUS ( ST_WIN_DFLT            ),
42     DEF_STATUS ( ST_BLOCK_OVERRIDE     )
43 };
44
45 #endif // _ERR_STAT
46
47
48 /* --- status codes 1000 and up are system- or module- specific ----- */
49
50 #endif // _STATUS_DEFINED
51

```

FIG. 18C

```

1  /* ----- */
2  /*          ETC: Global system definitions          */
3  /*          */
4  /*          IFACE.H - Interface support            */
5  /*          */
6  /*          Copyright (c) 1994-95 Object Dynamics Corp. All Rights Reserved. */
7  /* ----- */
8
9  #ifndef _IFACE_DEFINED
10 #define _IFACE_DEFINED
11
12 /* --- type definitions ----- */
13
14 typedef uint _iid;
15
16 #define NO_IID 0
17
18 #define IID_COMPAT(i1,i2) \
19     ((i1) == (i2) || \
20      (i1) == IID_I_POLY || \
21      (i2) == IID_I_POLY )
22
23 /* --- support macro definitions ----- */
24
25 class _COMPONENT; // forward
26
27 class _INTERFACE
28 {
29     public:
30         const _iid      iid;
31         const _COMPONENT *const objp;
32
33     public:
34         _INTERFACE (_iid id, _COMPONENT *op) : iid (id), objp (op) { }
35
36     protected:
37         virtual ~_INTERFACE (void) { } // (hack) - adds vtbl - simplifies ptr casts
38 #define __IFACE_OFFSET_OPO 1           // 1st operation starts at index 1 in vtbl
39     } ;                               // Keep in sync!
40
41 #define INTERFACE(i) \
42     class i : public _INTERFACE \
43     { \
44     public: \
45         i (_COMPONENT *op) : _INTERFACE (IID_##i, op) { } \
46     protected: \
47         i (void) : _INTERFACE (0, NULL) { } \
48         i (i& ir) : _INTERFACE (0, NULL) { } \
49         virtual ~i (void) { }
50
51 #define END_INTERFACE \
52     } ;
53
54 #define CONSTANTS      public:
55 #define BUSES          public:
56 #define OPERATIONS     public:
57 #define METHODS        public:
58
59 #define DEFINE_BUS \

```

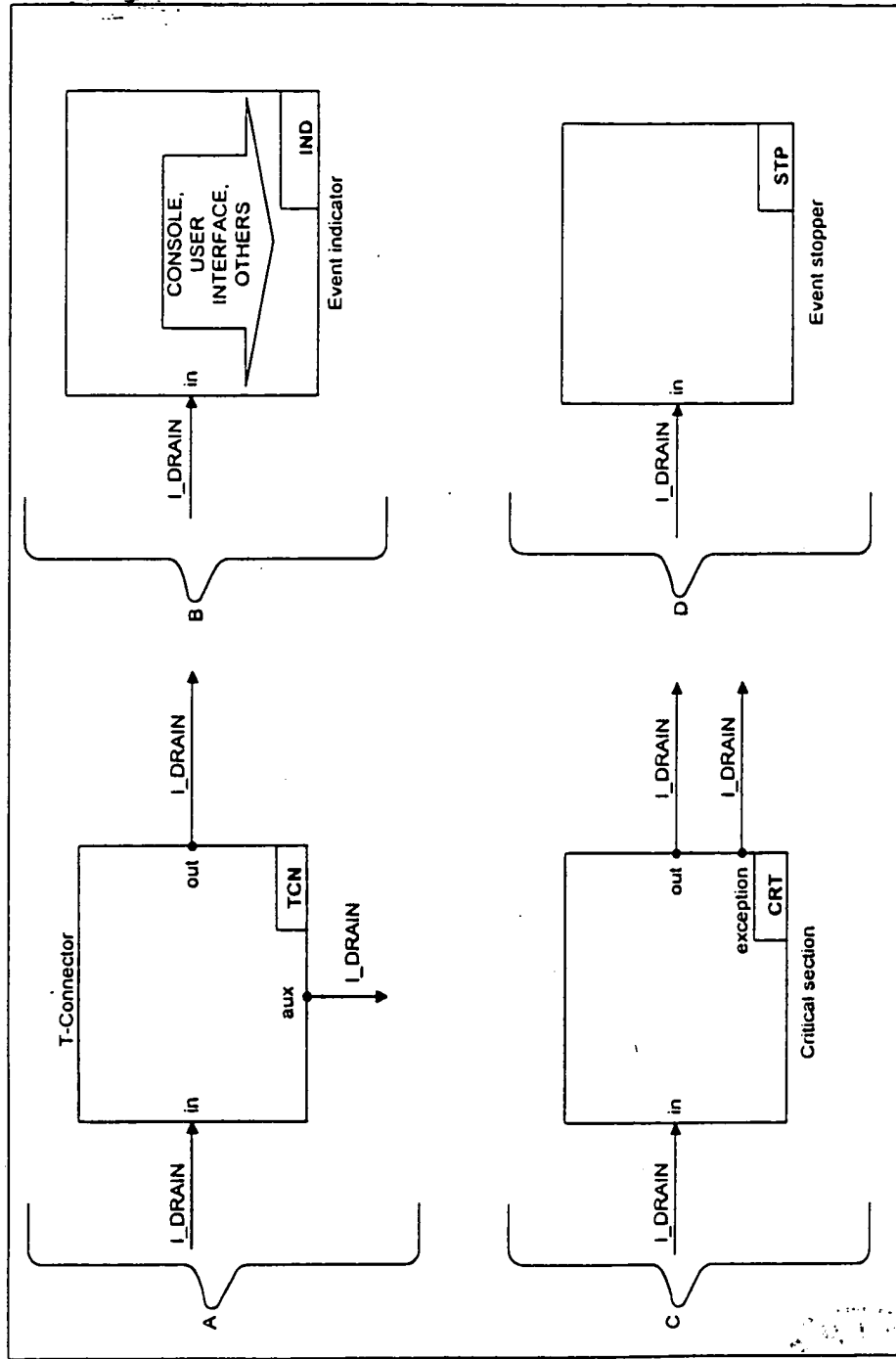



FIG. 31

```

1      struct BUS      \
2      {
3
4      #define BUS(x)    \
5          struct x      \
6          {
7
8      #define END_BUS    \
9          } ;
10
11     #define INTERFACE_BUS(x) \
12         typedef x BUS
13
14     #define DEFAULTS(x) \
15         BUS (void) { x; }
16
17     #define BUS_DEFAULTS(b,x) \
18         b (void) { x; }
19
20     #define FACTORY_DEFAULTS(x) \
21         FACTORY_BUS (void) { x; }
22
23     #define OP(o) \
24         virtual _stat o (BUS *bp) = 0
25
26     #endif // _IFACE_DEFINED

```

FIG. 19B

```

1  /* ----- */
2  /*               IFC: Project Interfaces               */
3  /* ----- */
4  /*               I_TERMINAL.H - Terminal control       */
5  /* ----- */
6  /*      Copyright (c) 1994-95 Object Dynamics Corp. All Rights Reserved.      */
7  /* ----- */
8
9  #ifndef _I_TERMINAL_DEFINED
10 #define _I_TERMINAL_DEFINED
11
12
13 BUS ( B_TERMINAL )
14     // terminal identification
15     char      *namep ; // terminal name
16     uint      name_sz; // size of name buffer
17
18     // terminal information
19     uint      type ; // terminal type (physical mechanism + direction)
20     uint32    card ; // terminal cardinality
21     flg16    attr ; // terminal attributes
22
23     // connection data
24     _iid      iid ; // terminal interface id (contract id)
25     _INTERFACE *ip ; // v-table interface
26
27     // connection identification
28     _id32     conn_id; // connection id (as defined by the terminal)
29     _COMPONENT *comp ; // pointer to component ($$EV_EVB, see notes)
30     _id       link_id; // link id ($$EV_EVB, see notes)
31
32 END_BUS
33
34 #define IID_I_TERMINAL    0x03
35
36 INTERFACE (I_TERMINAL)
37
38     /* ----- */
39     CONSTANTS
40         enum TYPE
41         {
42             T_INPUT  = 0,
43             T_OUTPUT = 1,
44         };
45
46         enum ATTR
47         {
48             A_STDSIG = (1UL << 14), // terminal for standard signal
49             A_FLOATING = (1UL << 15), // terminal is ok to be unconnected
50         };
51
52     /* ----- */
53     BUSES
54         INTERFACE_BUS ( B_TERMINAL );
55
56     /* ----- */
57     OPERATIONS
58         OP ( acquire      );
59         OP ( release      );

```

FIG 20A

```

1      OP ( connect      );
2      OP ( disconnect   );
3      OP ( get_info     );
4      OP ( get_context  );
5      OP ( set_context  );
6      OP ( qry_open     );
7      OP ( qry_get_first );
8      OP ( qry_get_last );
9      OP ( qry_get_next );
10     OP ( qry_get_prev  );
11     OP ( qry_get_curr  );
12     OP ( qry_close    );
13
14     END_INTERFACE
15
16     /* declaration macro */
17
18     #define DECLARE_I_TERMINAL(c,l) \
19         IDEF (c, l, I_TERMINAL) \
20             OPDECL ( acquire      ); \
21             OPDECL ( release     ); \
22             OPDECL ( connect     ); \
23             OPDECL ( disconnect  ); \
24             OPDECL ( get_info    ); \
25             OPDECL ( get_context ); \
26             OPDECL ( set_context ); \
27             OPDECL ( qry_open    ); \
28             OPDECL ( qry_get_first ); \
29             OPDECL ( qry_get_last ); \
30             OPDECL ( qry_get_next ); \
31             OPDECL ( qry_get_prev ); \
32             OPDECL ( qry_get_curr ); \
33             OPDECL ( qry_close   ); \
34     END_IDEF \
35     /* allow for semicolon */ \
36     typedef int c##_l##_DECLARE_I_TERMINAL_DUMMY
37
38     // Descriptions:
39
40     // on    acquire
41     // in :   namep   - terminal name
42     //       comp    - component on the other side
43     //       link_id - link id
44     // out: (ip)      - if type is T_INPUT - pointer to the terminal interface
45     //       type     - terminal type (I_TERMINAL::T_XXX)
46     //       iid      - expected interface id
47     //       conn_id  - connection id assigned for this connection
48     // act: acquire connection context
49     // s :   ST_NOT_FOUND - terminal not found
50     //       ST_NO_ROOM   - terminal cardinality exhausted
51
52     // on    release
53     // in :   namep   - terminal name
54     //       ip       - if type is T_OUTPUT - interface to which the output was
55     // connected
56     //                   if type is T_INPUT - interface to release
57     //                   ($$ may not be needed and in most cases can not be provided)
58     //       comp     - component on the other side
59     //       link_id  - link id

```

FIG 20B

```

1  //      conn_id - connection id assigned by this terminal on acquire
2  // out: void
3  // act: release terminal
4  // s : ST_NO_ACTION - terminal was not connected
5  //      (to/via specified interface or component)
6  //      ST_NOT_FOUND - terminal not found
7
8  // on connect
9  // in : namep - terminal name
10 //      type - terminal type (I_TERMINAL::T_xxx)
11 //      iid - expected interface id
12 //      (ip) - if type is T_OUTPUT - interface to connect to
13 //      comp - component to connect to
14 //      link_id - link id
15 //      conn_id - connection id assigned by this terminal on acquire
16 // out: void
17 // act: connect terminal
18 // s : ST_REFUSE - interface mismatch (e.g., unacceptable IID)
19 //      ST_NOT_FOUND - terminal not found
20
21 // on disconnect
22 // in : namep - terminal name
23 //      ip - if type is T_OUTPUT - interface to which the output is
24 connected
25 //      if type is T_INPUT - interface to disconnect
26 //      ($$ may not be needed and in most cases can not be provided)
27 //      comp - component to disconnect from
28 //      link_id - link id
29 //      conn_id - connection id assigned by this terminal on acquire
30 // out: void
31 // act: disconnect terminal
32 // s : ST_NO_ACTION - terminal was not connected
33 //      (to/via specified interface or component)
34
35 // on get_info
36 // in : namep - terminal name
37 // out: type - terminal type (I_TERMINAL::T_xxx)
38 //      iid - interface id of terminal
39 //      card - terminal cardinality (static, not current)
40 //      attr - terminal attributes
41 // act: return information about specified terminal
42 // s : ST_NOT_FOUND - terminal not found
43 // nb : the information returned by this operation is not affected by
44 //      the current state of the terminal
45
46 // on get_context
47 // not implemented
48
49 // on set_context
50 // not implemented
51
52 // on qry_open
53 // in : namep - query string
54 // out: ctx - query context for subsequent qry_xxx operations
55 // act: open query on terminal namespace
56 // s : ST_NO_ROOM - too many open queries
57 //      ST_BAD_SYNTAX - bad query syntax
58
59 // on qry_get_first

```

FIG 20C

```

1 // in : namep - buffer for name or NULL
2 //      (name_sz) - size of buffer, [bytes]
3 //      ctx - query context from previous qry_xxx operation
4 // out: (*namep) - terminal name
5 //      ctx - query context for subsequent qry_xxx operation
6 // act: get first matching terminal name
7 // s : ST_NOT_FOUND - no matching terminals
8
9 // on qry_get_last
10 // in : namep - buffer for name or NULL
11 //      (name_sz) - size of buffer, [bytes]
12 //      ctx - query context from previous qry_xxx operation
13 // out: (*namep) - terminal name
14 //      ctx - query context for subsequent qry_xxx operation
15 // act: get last matching terminal name
16 // s : ST_NOT_FOUND - no matching terminals
17
18 // on qry_get_next
19 // in : namep - buffer for name or NULL
20 //      (name_sz) - size of buffer, [bytes]
21 //      ctx - query context from previous qry_xxx operation
22 // out: (*namep) - terminal name
23 //      ctx - query context for subsequent qry_xxx operation
24 // act: get next matching terminal name
25 // s : ST_NOT_FOUND - no more matching terminals
26
27 // on qry_get_prev
28 // in : namep - buffer for name or NULL
29 //      (name_sz) - size of buffer, [bytes]
30 //      ctx - query context from previous qry_xxx operation
31 // out: (*namep) - terminal name
32 //      ctx - query context for subsequent qry_xxx operation
33 // act: get previous matching terminal name
34 // s : ST_NOT_FOUND - no more matching terminals
35
36 // on qry_get_curr
37 // in : namep - buffer for name or NULL
38 //      (name_sz) - size of buffer, [bytes]
39 //      ctx - query context from previous qry_xxx operation
40 // out: (*namep) - terminal name
41 // act: get current terminal name in query
42 // nb : ctx is unchanged
43
44 // on qry_close
45 // in : ctx - query context from qry_open or another qry_xxx operation
46 // out: void
47 // act: close query on terminal name space
48
49 // $SEV_EVB -- fields used by the event bus terminal interface
50 // Note: comp in the bus is provided for use by connectivity components only.
51 //      It should not be used by regular components & assemblies.
52 //      comp contains pointer identifying the component at the other side
53 //      of the link.
54 //      Both link_id and comp may be used for identification of the particular
55 //      connection on a terminal. They should not be interpreted in any way
56 //      nor the comp should be dereferenced. Sometimes the ip-field on
57 //      disconnect and release may be used for connection identification, too.
58
59 // Note: the assembly manager (dflt assembly implementation) does not store

```

FIG 20D

```
1 //      the connection id -- so it may not be provided on disconnect/release
2
3 // Note: $$ on disconnect ip may not be provided in many cases:
4 //      a) when output array is connected to input array
5 //      b) when assy output is connected to input array
6 //      In both examples above the input ptr cannot be obtained from the output
7 //      and therefore cannot be passed on input's disconnect.
8 //      (Generally, none of the existing parts uses ip for identification)
9
10 #endif // _I_TERMINAL_DEFINED
```

```

1  /* ----- */
2  /*          IFC: Project Interfaces          */
3  /* ----- */
4  /*          I_PROP.H - Property            */
5  /* ----- */
6  /*          Copyright (c) 1994-95 Object Dynamics Corp. All Rights Reserved. */
7  /* ----- */
8
9  #ifndef _I_PROP_DEFINED
10 #define _I_PROP_DEFINED
11
12 /* --- .type values */
13
14 #define PT_USER      0          // user defined
15 #define PT_INT       1          // signed int
16 #define PT_UINT      2          // unsigned int
17 #define PT_CHAR      3          // uchar
18 #define PT_UPCHAR    4          // upcaseduchar
19 #define PT_STRING    5          // asciiz, no multiple spaces
20 #define PT_NAME      6          // upcased STRING
21 #define PT_DATA      7          // data string (0x1b, '-0', etc.)
22 #define PT_NONE      8          // 1st invalid value of .type
23
24
25 BUS ( B_PROP )
26
27     char    *namep ;
28     size_t   name_sz;
29     uint32   type ;
30     void    *p ;
31     size_t   sz ;
32     size_t   len ;
33     _ctx     ctx ;
34
35 END_BUS
36
37 #define IID_I_PROP    0x02
38
39 INTERFACE ( I_PROP )
40
41     /* ----- */
42     CONSTANTS
43
44     /* ----- */
45     BUSES
46         INTERFACE_BUS ( B_PROP );
47
48     /* ----- */
49     OPERATIONS
50         OP ( get          );
51         OP ( set          );
52         OP ( chk          );
53         OP ( get_info     );
54         OP ( qry_open     );
55         OP ( qry_get_first );
56         OP ( qry_get_last  );
57         OP ( qry_get_next  );
58         OP ( qry_get_prev  );
59         OP ( qry_get_curr  );

```

FIG. 21A


```

1         OP ( qry_close      );
2
3     END_INTERFACE
4
5     /* declaration macro */
6
7     #define DECLARE_I_PROP(c,l)      \
8         #ifdef (c, l, I_PROP)      \
9             OPDECL ( get              ); \
10            OPDECL ( set              ); \
11            OPDECL ( chk              ); \
12            OPDECL ( get_info         ); \
13            OPDECL ( qry_open         ); \
14            OPDECL ( qry_get_first ); \
15            OPDECL ( qry_get_last  ); \
16            OPDECL ( qry_get_next  ); \
17            OPDECL ( qry_get_prev  ); \
18            OPDECL ( qry_get_curr  ); \
19            OPDECL ( qry_close      ); \
20        #endif
21        /* allow for semicolon */
22        typedef int c##_l##_DECLARE_I_PROP_DUMMY
23
24        // --- Descriptions
25
26        // note: on get, set and chk - if value type is specified and it doesn't match
27        the
28        //      internal type of the property, the component may (implementation-
29        defined)
30        //      convert the value to the appropriate type;
31
32        // on  get
33        // in : namep      - property name
34        //      p          - buffer for value or NULL
35        //      (sz)       - size of buffer (if p not NULL), [bytes]
36        //      type       - requested type or IP_NONE for any
37        // out:(*p)        - value
38        //      len        - value length (for both fixed & var size types), [bytes]
39        //      type       - property type
40        // act: get property value
41        // s  : ST_BAD_NAME   - property name not found
42        //      ST_OVERFLOW   - buffer too small
43        //      ST_REFUSE     - incorrect type
44        //      ST_BAD_ACCESS - property is write-only
45
46        // on  set
47        // in : namep      - property name
48        //      p          - buffer for value or NULL to clear
49        //      (len)      - size of value (for var size values), [bytes]
50        //      type       - type or IP_NONE if not known
51        // out: void
52        // act: set property value
53        // s  : ST_BAD_NAME   - property name not found
54        //      ST_BAD_VALUE  - bad property value
55        //      ST_OVERFLOW   - value too long
56        //      ST_REFUSE     - incorrect type
57        //      ST_BAD_ACCESS - property is read-only
58
59        // on  chk

```

FIG. 21B

```

1 // in : namep - property name
2 //      p      - buffer for value or NULL to clear
3 //      (len)   - size of value (for var size values), [bytes]
4 //      type    - value type or IP_NONE if not known
5 // out: void
6 // act: check if property may be set to value (valchk)
7 // s : ST_BAD_NAME - property name not found
8 //      ST_BAD_VALUE - bad property value
9 //      ST_OVERFLOW - value too long
10 //      ST_REFUSE - incorrect type
11 //      ST_BAD_ACCESS - property is read-only
12
13
14 // on get_info
15 // in : namep - property name
16 // out: type - property type
17 // act: get information about specified property
18 // s : ST_BAD_NAME - property name not found
19 // nb : the information returned by this operation is not affected by
20 //      the current value of the property
21
22 // on qry_open
23 // in : namep - query string
24 // out: ctx - query context for subsequent qry_xxx operations
25 // act: open query on property space
26 // s : ST_NO_ROOM - too many open queries
27 //      ST_BAD_SYNTAX - bad query syntax
28
29 // on qry_get_first
30 // in : namep - buffer for name or NULL
31 //      (name_sz) - size of buffer, [bytes]
32 //      ctx - query context from previous qry_xxx operation
33 // out: (*namep) - property name
34 //      ctx - query context for subsequent qry_xxx operation
35 // act: get first matching property name
36 // s : ST_NOT_FOUND - no matching properties
37
38 // on qry_get_last
39 // in : namep - buffer for name or NULL
40 //      (name_sz) - size of buffer, [bytes]
41 //      ctx - query context from previous qry_xxx operation
42 // out: (*namep) - property name
43 //      ctx - query context for subsequent qry_xxx operation
44 // act: get last matching property name
45 // s : ST_NOT_FOUND - no matching properties
46
47 // on qry_get_next
48 // in : namep - buffer for name or NULL
49 //      (name_sz) - size of buffer, [bytes]
50 //      ctx - query context from previous qry_xxx operation
51 // out: (*namep) - property name
52 //      ctx - query context for subsequent qry_xxx operation
53 // act: get next matching property name
54 // s : ST_NOT_FOUND - no more matching properties
55
56 // on qry_get_prev
57 // in : namep - buffer for name or NULL
58 //      (name_sz) - size of buffer, [bytes]
59 //      ctx - query context from previous qry_xxx operation

```

FIG. 21C

```

1 // out: (*namep) - property name
2 //      ctx      - query context for subsequent qry_xxx operation
3 // act: get previous matching property name
4 // s - : _ST_NOT_FOUND - no more matching properties
5
6 // on  qry_get_curr
7 // in : namep      - buffer for name or NULL
8 //      (name_sz) - size of buffer, [bytes]
9 //      ctx        - query context from previous qry_xxx operation
10 // out: (*namep) - property name
11 // act: get current property name in query
12 // nb : ctx is unchanged
13
14 // on  qry_close
15 // in : ctx      - query context from qry_open or another qry_xxx operation
16 // out: void
17 // act: close query on property space
18
19 #endif // _I_PROP_DEFINED
20

```

FIG. 21D

```

1  /* ----- */
2  /*          IFC: Project Interfaces          */
3  /* ----- */
4  /*          I_FACTORY.H - Component creation / destruction      */
5  /* ----- */
6  /*          Copyright (c) 1994-95 Object Dynamics Corp. All Rights Reserved. */
7  /* ----- */
8
9  #ifndef _I_FACTORY_DEFINED
10 #define _I_FACTORY_DEFINED
11
12
13 #define IID_I_FACTORY    0x01
14
15 INTERFACE (I_FACTORY)
16
17 /* ----- */
18 CONSTANTS
19     enum
20     {
21         A_PRIMARY    = (1UL << 14),    // create primary object (factory)
22         A_NO_ALLOC    = (1UL << 13),    // don't call new to alloc memory
23         A_GENERIC     = (1UL << 12),    // generic factory bus (no ext. data)
24     };
25
26 /* ----- */
27 BUSES
28     DEFINE_BUS
29         flg32                attr;        // factory attributes
30         _COMPONENT            *objp;       // ptr to component or memory
31         _INTERFACE            *term;       // terminal interface
32         _INTERFACE            *prop;       // property interface
33         _ctx                  ctx ;        // creation context or NO_CTX
34     END_BUS
35
36 /* ----- */
37 OPERATIONS
38     OP ( create_dflts );
39     OP ( create );
40     OP ( destroy );
41     OP ( get_part_ifc );
42
43 END_INTERFACE
44
45 /* declaration macro */
46
47 #define DECLARE_I_FACTORY(c,l)          \
48     IDEF (c, 1, I_FACTORY)              \
49     OPDECL (create ) ;                  \
50     OPDECL (create_dflts ) ;            \
51     OPDECL (destroy ) ;                 \
52     OPDECL (get_part_ifc ) ;            \
53     END_IDEF                            \
54     /* allow for semicolon */          \
55     typedef int c##_l##_DECLARE_I_FACTORY_DUMMY
56
57 /* --- Descriptions ----- */
58
59 // on    create_dflts    (struct BUS or derived)

```

FIG. 22A

```

1 // in : void
2 // out: attr - default attributes
3 //      objp - NULL
4 // act: fill FACTORY_BUS with defaults
5
6 // on   create      (struct BUS or derived)
7 // in : attr - object attributes (A_xxx)
8 //      (objp) - ptr to allocated memory for object instance (if A_NO_ALLOC)
9 //      ctx  - additional context or NO_CTX
10 // out: objp - ptr to new object
11 //      temp - pointer to the terminal interface of the part
12 //      propp - pointer to the property interface of the part
13 // act: create new object
14 // nb : if A_GENERIC is set, then the bus passed is strictly I_FACTORY::BUS and
15 //      not a derived one.
16 // nb : ctx=NO_CTX is always accepted; some components may allow additional
17 //      parameters in/through this field
18
19 // on   destroy      (struct BUS or derived)
20 // in : void
21 // out: void
22 // act: destroy object
23
24 // on   get_part_ifc (struct BUS or derived)
25 // in : void
26 // out: temp - pointer to the terminal interface of the part
27 //      propp - pointer to the property interface of the part
28 // act: return part interfaces
29
30 #endif // _I_FACTORY_DEFINED

```

FIG. 22B

```

1  /* ----- */
2  /*          CLP: Collections Parts          */
3  /* ----- */
4  /*          CP_BBA.CPP - Bag on Byte Array  */
5  /* ----- */
6  /* Copyright (c) 1993-1994 Object Dynamics Corp. All Rights Reserved. */
7  /* ----- */
8
9  #include <rtx_pre.h>
10 #include <stddef.h>
11 #include <stdlib.h>
12 #include <memory.h>
13 #include <rtx_post.h>
14
15 #include <rtx.h>
16 #include <asm.h>
17 #include <thp.h>
18
19 #include <iface.h>
20 #include <part.h>
21
22 #include <i_factor.h>
23 #include <i_prop.h>
24 #include <i_drain.h>
25 #include <i_bag.h>
26 #include <i_agm.h>
27 #include <i_bytear.h>
28
29 /* --- Definitions ----- */
30
31 struct CP_BBA_ENTRY_HDR
32 {
33     enum ATTR
34     {
35         A_VALID      = VALID,
36         A_FREE       = FREE,
37         A_KILLED     = KILLED,
38     };
39
40     flg32      attr;          // attributes [A_VALID, A_FREE, A_KILLED]
41     uint32     card;         // cardinality
42 };
43
44 /** --- Component Definition ----- */
45
46 #define CP_BBA_NAME "CP_BBA"
47
48 PART ( CP_BBA )
49
50
51 /* ----- */
52 BUSES
53     DEFAULT_FACTORY_BUS ( CP_BBA );
54
55 /* ----- */
56 INPUTS
57     IDECL ( CP_BBA , factory , I_FACTORY ) ;
58     IDECL ( CP_BBA , term   , I_TERMINAL ) ;
59     IDECL ( CP_BBA , prop   , I_PROP   ) ;

```

FIG. 23A

```

1      IDECL ( CP_BBA , bag          , I_BAG          );
2
3      IDECL_RST (CP_BBA);
4
5      OUTPUTS
6      ODECL ( CP_BBS , arr          , I_BYTEARR      );
7
8
9      END_PART // CP_BBA
10
11     #ifdef BIT32
12         #define hmemcpy memcpy
13         #define HUGE
14     #else
15         #define HUGE __huge
16     #endif
17
18     /* --- Definitions ----- */
19
20     #define AGM_CHUNK      1024
21
22     BEGIN_SELF ( CP_BBA )
23         // data
24         uint32      n_entries;    // number of entries
25         void        *bufp;        // buffer for one entry
26         uint32      buff_sz;      // buffer size (dep. on data_sz)
27
28         // properties
29         uint32      data_sz;       // data size
30         uint32      key_sz;        // key size
31         uint32      key_off;       // key offset
32         char        name [64];
33     END_SELF
34
35     BEGIN_CLASS_DATA ( CP_BBA )
36         uint16      instance_counter;    // ...
37     END_CLASS_DATA
38
39     /* --- Interface declarations ----- */
40
41     DECLARE_I_FACTORY ( CP_BBA, factory );
42     DECLARE_I_TERMINAL ( CP_BBA, term );
43     DECLARE_I_PROP ( CP_BBA, prop );
44     DECLARE_I_BAG ( CP_BBA, bag );
45
46     DECLARE_RST (CP_BBA);
47
48     /* --- I/O tables ----- */
49
50     INPUT_TABLE
51         INPUT (CP_BBA, prop , I_PROP )
52         INPUT (CP_BBA, bag , I_BAG )
53
54         INPUT_RST (CP_BBA)
55     END_INPUT_TABLE
56
57     OUTPUT_TABLE
58         OUTPUT (CP_BBA, arr , I_BYTEARR )
59

```

FIG. 23B

```

1  END_OUTPUT_TABLE
2
3  /* --- Constants & tables ----- */
4
5  PROPERTIES
6      /* type      name      base field      min      max      */
7
8      PROP_UINT    ("DATA_SIZE"      , SELF, data_sz , 1 , 1024 ),
9      PROP_UINT    ("KEY_SIZE"       , SELF, key_sz  , 1 , 1024 ),
10     PROP_UINT    ("KEY_OFFSET"      , SELF, key_off , 0 , 1023 ),
11
12     PROP_STRING  ("name"            , SELF, name   , 0 , 64   ),
13 END_PROPERTIES
14
15 /* --- Init / Cleanup ----- */
16
17 INIT (CP_BBA)
18 {
19     /** init class data */
20     cp->instance_counter = 0;
21
22     REGISTER (CP_BBA);
23
24     return (ST_OK);
25 } /* INIT (CP_BBA) */
26
27 CLEANUP (CP_BBA)
28 {
29     Deregister (CP_BBA);
30
31     return (ST_OK);
32 } /* CLEANUP (CP_BBA) */
33
34 /* --- Constructors ----- */
35
36 DEFAULT_PART_CONSTRUCTORS (CP_BBA);
37
38
39 /* --- Operations: factory ----- */
40
41 DEFAULT_PART_FACTORY (CP_BBA, factory);
42 DEFAULT_PART_TERMINAL (CP_BBA, term );
43 DEFAULT_PART_PROP     (CP_BBA, prop );
44
45 /* --- Operations ----- */
46
47 METHOD (CP_BBA, create_dflts , FACTORY_BUS)
48 {
49     // ...
50
51     return (ST_OK);
52 } END_METHOD /* CP_BBA, create_dflts */
53
54 METHOD (CP_BBA, create , FACTORY_BUS)
55 {
56     _stat s;
57
58     /** valchk */
59

```

FIG. 23C


```

1      if (bp == NULL) return (ST_NULL_PTR);
2
3      /** open interfaces */
4      OPEN_INTERFACE (CP_BBA, factory    );
5      if (bp->attr & I_FACTORY::A_PRIMARY) return (ST_OK);    // valid primary obj
6      OPEN_INTERFACE (CP_BBA, term      );
7      OPEN_INTERFACE (CP_BBA, prop      );
8      OPEN_INTERFACE (CP_BBA, bag       );
9      OPEN_RST       (CP_BBA);
10
11     /** init self */
12     sp->n_entries = 0;
13     sp->bufp      = NULL;
14     sp->buff_sz   = 0;
15     sp->data_sz   = 2;
16     sp->key_sz    = 2;
17     sp->key_off   = 0;
18
19     /** internal reset */
20     (void)call (sp->>thisp->iRST, raise, NULL);
21
22     /** special processing for default name */
23     os_con_sprintf (sp->name, "%50s%d", CP_BBA_NAME, ++(cp->instance_counter));
24
25     return (ST_OK);
26     } END_METHOD /* CP_BBA, create */
27
28 METHOD (CP_BBA, destroy, FACTORY_BUS)
29 {
30     /** close interfaces */
31     CLOSE_INTERFACE (CP_BBA, factory    );
32     CLOSE_INTERFACE (CP_BBA, term      );
33     CLOSE_INTERFACE (CP_BBA, prop      );
34     CLOSE_INTERFACE (CP_BBA, bag       );
35     CLOSE_RST       (CP_BBA);
36
37     /** free entry buffer */
38     if (sp->bufp != NULL) (void)os_smp_free (sp->bufp);
39
40     /** init self */
41     sp->n_entries = 0;
42     sp->bufp      = NULL;
43     sp->buff_sz   = 0;
44     sp->data_sz   = 2;
45
46     return (ST_OK);
47     } END_METHOD /* CP_BBA, destroy */
48
49 /* --- Operations: bag ----- */
50
51 OPERATION ( CP_BBA, bag, add      )
52 {
53     I_BYTEARR::BUS      ab;
54     uint32              i;
55     CP_BBA_ENTRY_HDR    *hdrp;
56     byte                *datap;
57     _stat                s;
58
59     /** valchk */

```

FIG. 23D

```

1  if (bp == NULL) return (ST_NULL_PTR);
2  if (bp->datap == NULL) return (ST_INVALID);
3
4  /** Internal init */
5  hdrp = (CP_BBA_ENTRY_HDR *)sp->bufp;
6  datap = (byte *)sp->bufp + sizeof (*hdrp);
7
8  /** --- lookup */
9  for (i = 0; i < sp->n_entries; i++)
10 {
11     /** read entry */
12     ab.p = sp->bufp;
13     ab.sz = sp->buff_sz;
14     ab.pos = i * sp->buff_sz;
15     ab.len = sp->buff_sz;
16     ab.attr = I_BYTEARR::A_EXACT;
17     s = out (arr, read, &ab);
18     if (s == ST_EOF) return (ST_UNEXPECTED);
19     if_ret (s, s);
20
21     /** skip if empty */
22     if ((hdrp->attr == CP_BBA_ENTRY_HDR::A_FREE) ||
23         (hdrp->attr == CP_BBA_ENTRY_HDR::A_KILLED)) continue;
24
25     /** skip if different */
26     if (memcmp ((void *) (datap + sp->key_off),
27                 (void *) ((byte *)bp->datap + sp->key_off),
28                 sp->key_sz) != 0) continue;
29
30     /** sanity chk */
31     if (hdrp->card == 0) return (ST_UNEXPECTED);
32
33     /** increment cardinality */
34     if (hdrp->card < MAXUINT32 - 1) (hdrp->card)++;
35     else return (ST_OVERFLOW);
36
37     /** write back header only */
38     ab.p = hdrp;
39     ab.len = sizeof (*hdrp);
40     ab.pos = i * sp->buff_sz;
41     ab.attr = 0;
42     s = out (arr, write, &ab);
43     if (s == ST_OVERFLOW) return (ST_UNEXPECTED);
44     if_ret (s, s);
45
46     /** pass cardinality */
47     bp->card = hdrp->card;
48
49     /** return */
50     return (ST_OK);
51 } // for
52
53 /** --- add */
54 for (i = 0; i < sp->n_entries; i++)
55 {
56     /** read entry */
57     ab.p = sp->bufp;
58     ab.sz = sp->buff_sz;
59     ab.pos = i * sp->buff_sz;

```

FIG. 23E

```

1      ab.len = sp->buff_sz;
2      ab.attr = I_BYTEARR::A_EXACT;
3      s = out (arr, read, &ab);
4      if (s == ST_EOF) return (ST_UNEXPECTED);
5      if_ret (s, s);
6
7      /** write if empty */
8      if ((hdrp->attr == CP_BBA_ENTRY_HDR::A_FREE) != 0)
9      {
10         /** init header */
11         hdrp->card = 1;
12         hdrp->attr = CP_BBA_ENTRY_HDR::A_VALID;
13
14         /** init data */
15         memcpy ((void *)datap, bp->datap, sp->data_sz);
16
17         /** write header and data */
18         ab.p = sp->buffp;
19         ab.len = sp->buff_sz;
20         ab.pos = i * sp->buff_sz;
21         ab.attr = 0;
22         s = out (arr, write, &ab);
23         if (s == ST_OVERFLOW) return (ST_UNEXPECTED);
24         if_ret (s, s);
25
26         /** pass cardinality */
27         bp->card = hdrp->card;
28
29         /** return */
30         return (ST_OK);
31     }
32 } // for
33
34 /** chk for overflow */
35 if (sp->n_entries >= MAXUINT32 - 1) return (ST_NO_ROOM);
36
37 /** init header */
38 hdrp->card = 1;
39 hdrp->attr = CP_BBA_ENTRY_HDR::A_VALID;
40
41 /** init data */
42 memcpy ((void *)datap, bp->datap, sp->data_sz);
43
44 /** append */
45 ab.p = sp->buffp;
46 ab.len = sp->buff_sz;
47 ab.pos = sp->n_entries * sp->buff_sz;
48 ab.attr = I_BYTEARR::A_GROW;
49 s = out (arr, write, &ab);
50 if (s == ST_OVERFLOW) return (ST_UNEXPECTED);
51 if_ret (s, s);
52
53 /** increment number of entries */
54 (sp->n_entries)++;
55
56 /** pass cardinality */
57 bp->card = hdrp->card;
58
59 return (ST_OK);

```

FIG. 23F

```

1      ) END_OPERATION /* bag, add      */
2
3      // TO DO: other operation of I_BAG to be implemented here_
4
5      /* --- Signals ----- */
6
7      SIGNAL_RST ( CP_BBA )
8      {
9          I_AGM::BUS      ab;
10         _stat          s;
11
12         /** free entry buffer */
13         if (sp->bufp != NULL) (void)os_smp_free (sp->bufp);
14
15         /** init self */
16         sp->n_entries = 0;
17         sp->bufp      = NULL;
18         sp->buff_sz   = sizeof (CP_BBA_ENTRY_HDR) + max (sp->data_sz, 1);
19
20         /** alloc entry buffer */
21         if_ret (s, os_smp_alloc (sp->buff_sz, &(sp->bufp)));
22
23         return (ST_OK);
24     } END_SIGNAL /* RST */
25
26

```

FIG. 23G

```

1  /* ----- */
2  /*                      EVP: Event Flow Parts                      */
3  /* ----- */
4  /*                      EV_PCN.CPP - POLY T-Connector              */
5  /* ----- */
6  /*      Copyright (c) 1994-95 Object Dynamics Corp. All Rights Reserved. */
7  /* ----- */
8  /* Version 1.10 */
9  /* ----- */
10
11 #include <rtx_pre.h>
12 #include <stddef.h>
13 #include <stdlib.h>
14 #include <rtx_post.h>
15
16 #include <rtx.h>
17 #include <asm.h>
18
19 #include <iface.h>
20 #include <part.h>
21 #include <assembly.h>
22
23 #include <i_factor.h>
24 #include <i_prop.h>
25 #include <i_term.h>
26 #include <i_drain.h>
27 #include <i_poly.h>
28
29 #define EV_PCN_NAME "EV_PCN"
30
31 ASSEMBLY (EV_PCN)
32
33 /* ----- */
34 BUSES
35     DEFAULT_FACTORY_BUS (EV_PCN);
36
37 /* ----- */
38 INPUTS
39     IDECL ( EV_PCN , factory      , I_FACTORY );
40     IDECL ( EV_PCN , term        , I_TERMINAL );
41     IDECL ( EV_PCN , termi       , I_TERMINAL );
42     IDECL ( EV_PCN , prop        , I_PROP );
43     IDECL ( EV_PCN , propi       , I_PROP );
44     IDECL ( EV_PCN , in         , I_POLY );
45
46 OUTPUTS
47     ODECL ( EV_PCN , out         , I_POLY );
48     ODECL ( EV_PCN , aux         , I_DRAIN );
49
50
51 END_ASSEMBLY
52
53
54 /* --- Definitions ----- */
55
56 BEGIN_SELF (EV_PCN)
57
58     DEFAULT_ASM_SELF;
59

```

FIG. 24A

```

1      /* group properties */
2      uint32      ev_id_min;
3      uint32      ev_id_max;
4
5      END_SELF
6
7      BEGIN_CLASS_DATA (EV_PCN)
8          // none
9      END_CLASS_DATA
10
11     DECLARE_ASM (EV_PCN);
12
13     /* --- I/O tables ----- */
14
15     INPUT_TABLE
16         INPUT      (EV_PCN, prop      , I_PROP      )
17         INPUT      (EV_PCN, in        , I_POLY      )
18     END_INPUT_TABLE
19
20     OUTPUT_TABLE
21         OUTPUT_I   (EV_PCN, out       , I_POLY      )
22         OUTPUT_I   (EV_PCN, aux       , I_DRAIN     )
23     END_OUTPUT_TABLE
24
25     /* --- Constants & tables ----- */
26
27     ASM_PARTS
28         NEW_PART ( p2d, EV_P2D )
29         NEW_PART ( tcn, EV_TCN )
30         NEW_PART ( d2p, EV_D2P )
31     END_ASM_PARTS
32
33     ASM_PARAMETRIZATION
34         // none
35     END_ASM_PARAMETRIZATION
36
37     ASM_LINKS
38         //          from output          to input
39
40         LINK      ( *      , in          , p2d, in      )
41         LINK      ( p2d, out          , tcn, in      )
42         LINK      ( tcn, out          , d2p, in      )
43         LINK      ( d2p, out          , *      , out      )
44         LINK      ( tcn, aux          , *      , aux      )
45
46     END_ASM_LINKS
47
48     ASM_PROPERTY_GROUP ( ev_id_min_group )
49         ASM_PROP_MEMBER ( tcn      , OUT.EV_ID_MIN )
50         ASM_PROP_MEMBER ( tcn      , AUX.EV_ID_MIN )
51     END_ASM_PROPERTY_GROUP
52
53     ASM_PROPERTY_GROUP ( ev_id_max_group )
54         ASM_PROP_MEMBER ( tcn      , OUT.EV_ID_MAX )
55         ASM_PROP_MEMBER ( tcn      , AUX.EV_ID_MAX )
56     END_ASM_PROPERTY_GROUP
57
58     ASM_PROPERTIES
59         ASM_PROP_ALIAS ( OUT.INVERT      , tcn , OUT.INVERT )

```

FIG. 24B

```

1      ASM_PROP_ALIAS ( AUX.INVERT      , tcn , AUX.INVERT      )
2
3      ASM_PROP_GROUP ( EV_ID_MIN       , ev_id_min_group      )
4      ASM_PROP_GROUP ( EV_ID_MAX       , ev_id_max_group      )
5  END_ASM_PROPERTIES
6
7  ASM_GROUP_PROPERTY_DESC
8
9      /* type      name      base field      min      max      */
10
11      PROP_UINT ("EV_ID_MIN", SELF, ev_id_min , MINUINT32 , MAXUINT32 ),
12      PROP_UINT ("EV_ID_MAX", SELF, ev_id_max , MINUINT32 , MAXUINT32 ),
13
14  END_ASM_GROUP_PROPERTY_DESC
15
16  DEFAULT_ASM (EV_PCN) ;
17

```

FIG. 24C


```

1      OP ( op33      );
2      OP ( op34      );
3      OP ( op35      );
4      OP ( op36      );
5      OP ( op37      );
6      OP ( op38      );
7      OP ( op39      );
8      OP ( op40      );
9      OP ( op41      );
10     OP ( op42      );
11     OP ( op43      );
12     OP ( op44      );
13     OP ( op45      );
14     OP ( op46      );
15     OP ( op47      );
16     OP ( op48      );
17     OP ( op49      );
18     OP ( op50      );
19     OP ( op51      );
20     OP ( op52      );
21     OP ( op53      );
22     OP ( op54      );
23     OP ( op55      );
24     OP ( op56      );
25     OP ( op57      );
26     OP ( op58      );
27     OP ( op59      );
28     OP ( op60      );
29     OP ( op61      );
30     OP ( op62      );
31     OP ( op63      );
32
33     END_INTERFACE
34
35     /* declaration macro */
36
37     #define DECLARE_I_POLY(c,l)      \
38         IDEF (c, l, I_POLY)          \
39         OPDECL ( op0      ) ;        \
40         OPDECL ( op1      ) ;        \
41         OPDECL ( op2      ) ;        \
42         OPDECL ( op3      ) ;        \
43         OPDECL ( op4      ) ;        \
44         OPDECL ( op5      ) ;        \
45         OPDECL ( op6      ) ;        \
46         OPDECL ( op7      ) ;        \
47         OPDECL ( op8      ) ;        \
48         OPDECL ( op9      ) ;        \
49         OPDECL ( op10     ) ;        \
50         OPDECL ( op11     ) ;        \
51         OPDECL ( op12     ) ;        \
52         OPDECL ( op13     ) ;        \
53         OPDECL ( op14     ) ;        \
54         OPDECL ( op15     ) ;        \
55         OPDECL ( op16     ) ;        \
56         OPDECL ( op17     ) ;        \
57         OPDECL ( op18     ) ;        \
58         OPDECL ( op19     ) ;        \
59         OPDECL ( op20     ) ;

```

FIG. 25B

```

1      OPDECL ( op21      ) ;      \
2      OPDECL ( op22      ) ;      \
3      OPDECL ( op23      ) ;      \
4      OPDECL ( op24      ) ;      \
5      OPDECL ( op25      ) ;      \
6      OPDECL ( op26      ) ;      \
7      OPDECL ( op27      ) ;      \
8      OPDECL ( op28      ) ;      \
9      OPDECL ( op29      ) ;      \
10     OPDECL ( op30      ) ;      \
11     OPDECL ( op31      ) ;      \
12     OPDECL ( op32      ) ;      \
13     OPDECL ( op33      ) ;      \
14     OPDECL ( op34      ) ;      \
15     OPDECL ( op35      ) ;      \
16     OPDECL ( op36      ) ;      \
17     OPDECL ( op37      ) ;      \
18     OPDECL ( op38      ) ;      \
19     OPDECL ( op39      ) ;      \
20     OPDECL ( op40      ) ;      \
21     OPDECL ( op41      ) ;      \
22     OPDECL ( op42      ) ;      \
23     OPDECL ( op43      ) ;      \
24     OPDECL ( op44      ) ;      \
25     OPDECL ( op45      ) ;      \
26     OPDECL ( op46      ) ;      \
27     OPDECL ( op47      ) ;      \
28     OPDECL ( op48      ) ;      \
29     OPDECL ( op49      ) ;      \
30     OPDECL ( op50      ) ;      \
31     OPDECL ( op51      ) ;      \
32     OPDECL ( op52      ) ;      \
33     OPDECL ( op53      ) ;      \
34     OPDECL ( op54      ) ;      \
35     OPDECL ( op55      ) ;      \
36     OPDECL ( op56      ) ;      \
37     OPDECL ( op57      ) ;      \
38     OPDECL ( op58      ) ;      \
39     OPDECL ( op59      ) ;      \
40     OPDECL ( op60      ) ;      \
41     OPDECL ( op61      ) ;      \
42     OPDECL ( op62      ) ;      \
43     OPDECL ( op63      ) ;      \
44     END_IDEF
45     /* allow for semicolon */
46     typedef int c##_1##_DECLARE_I_POLY_DUMMY
47
48
49 #endif // _I_POLY_DEFINED
50

```

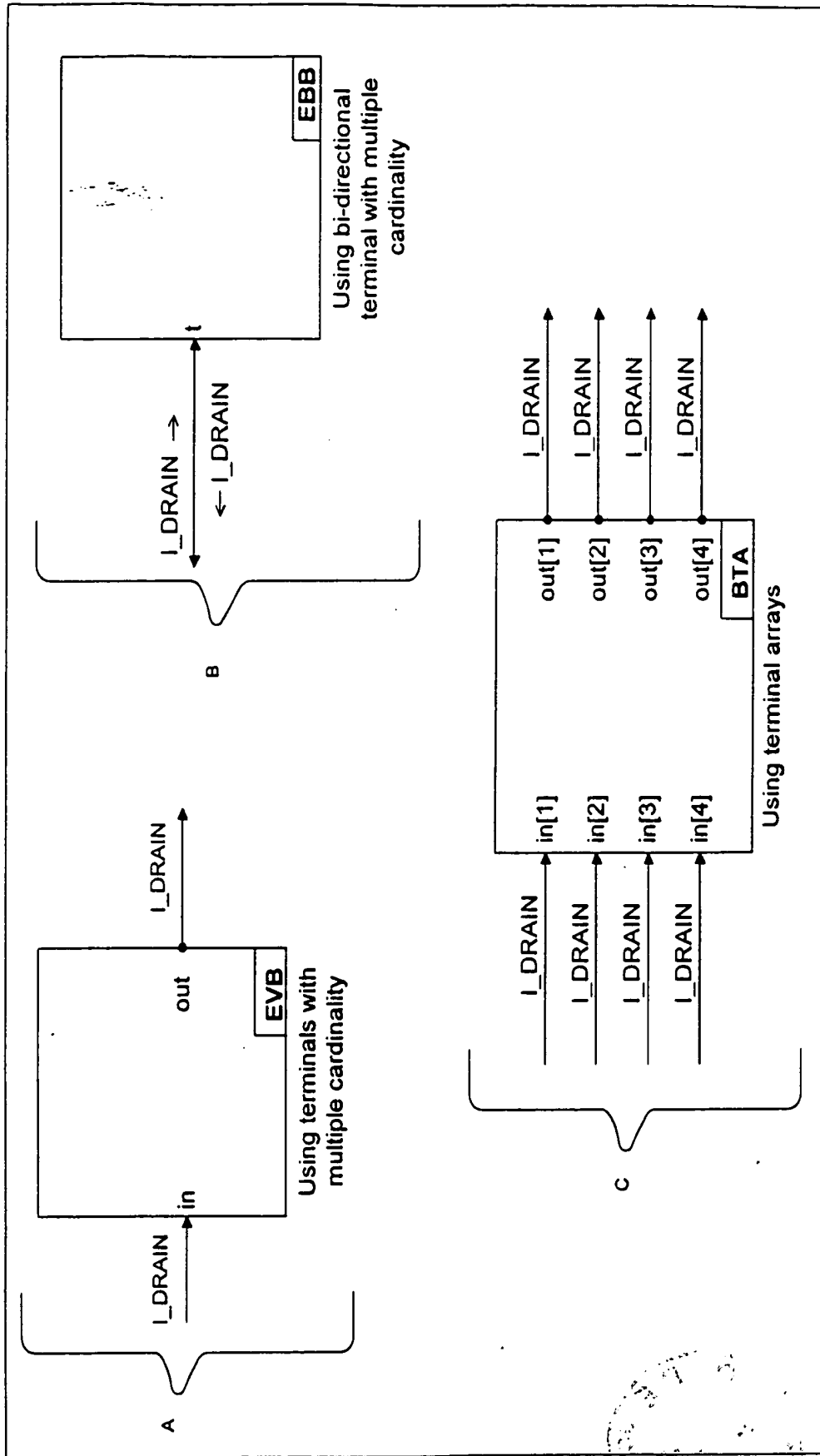


FIG. 26

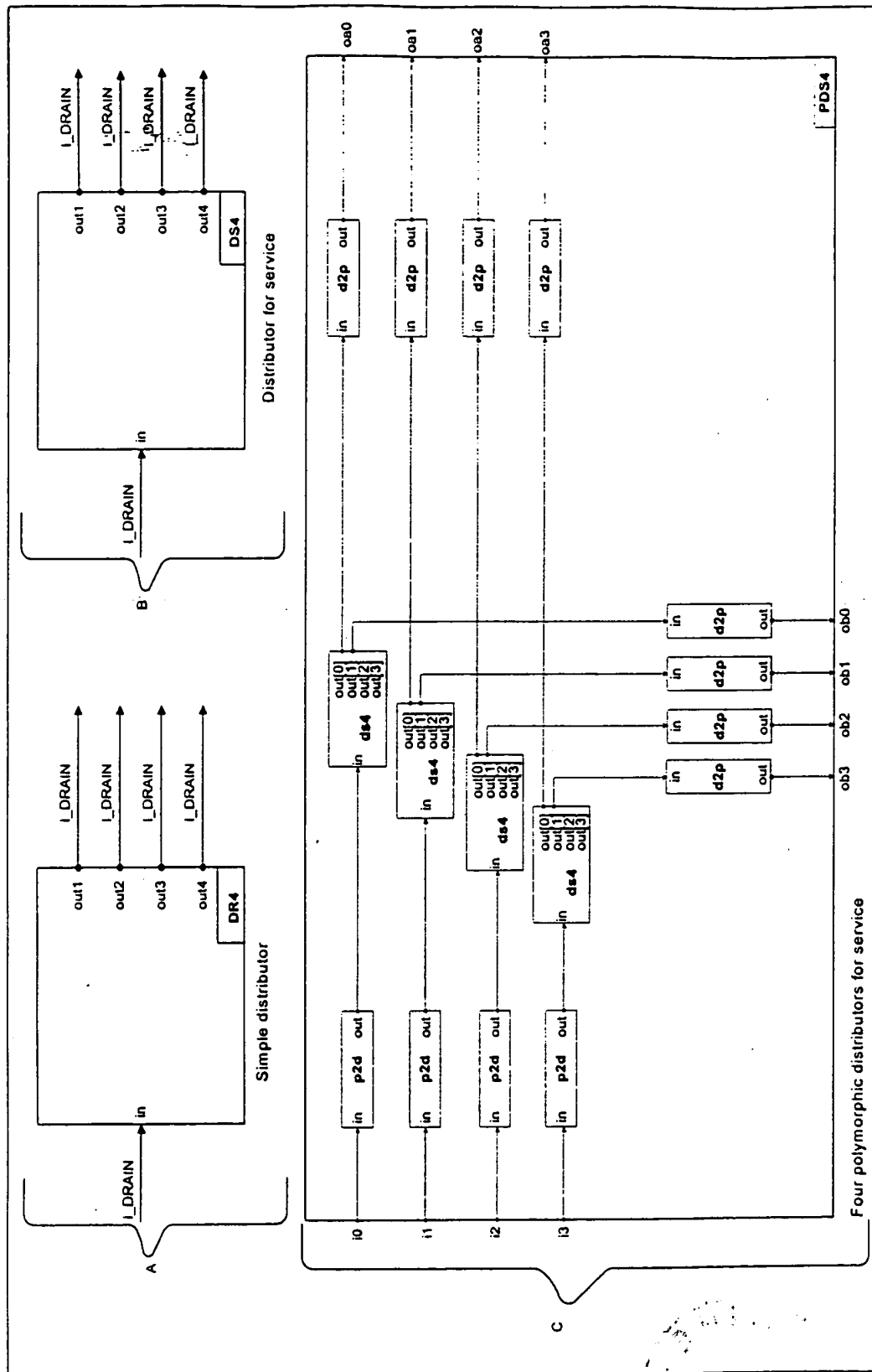


FIG. 27

104090 6920860

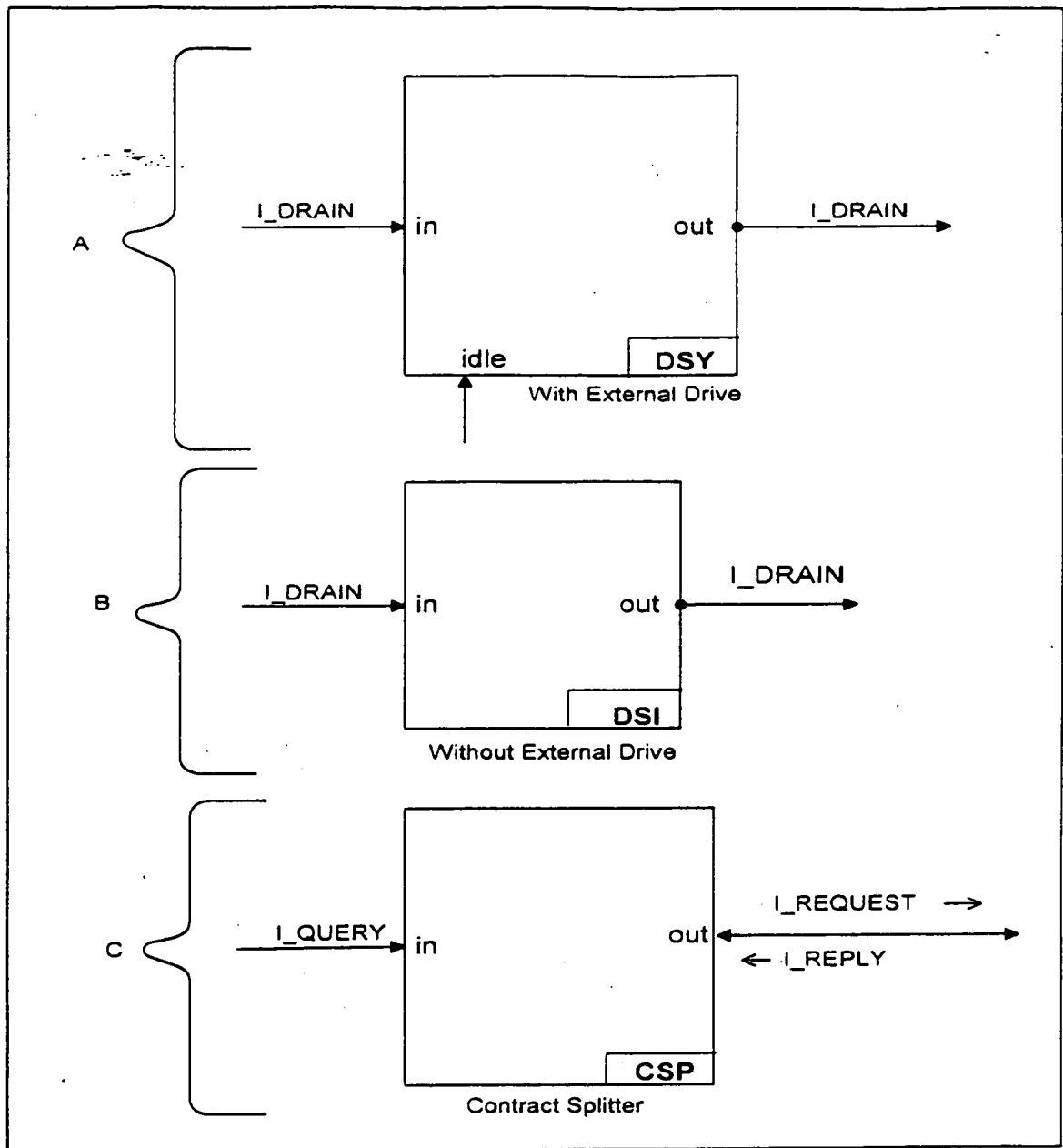


FIG. 28

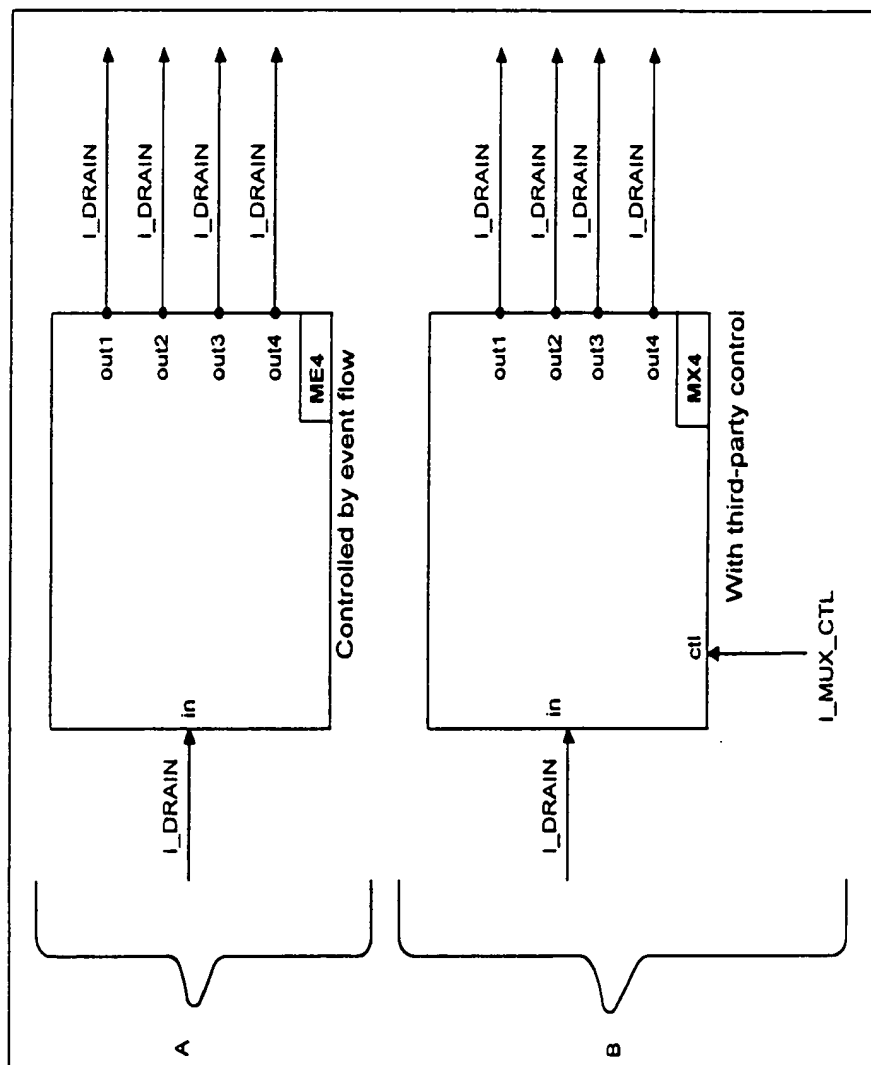


FIG. 29

06802697-0604401

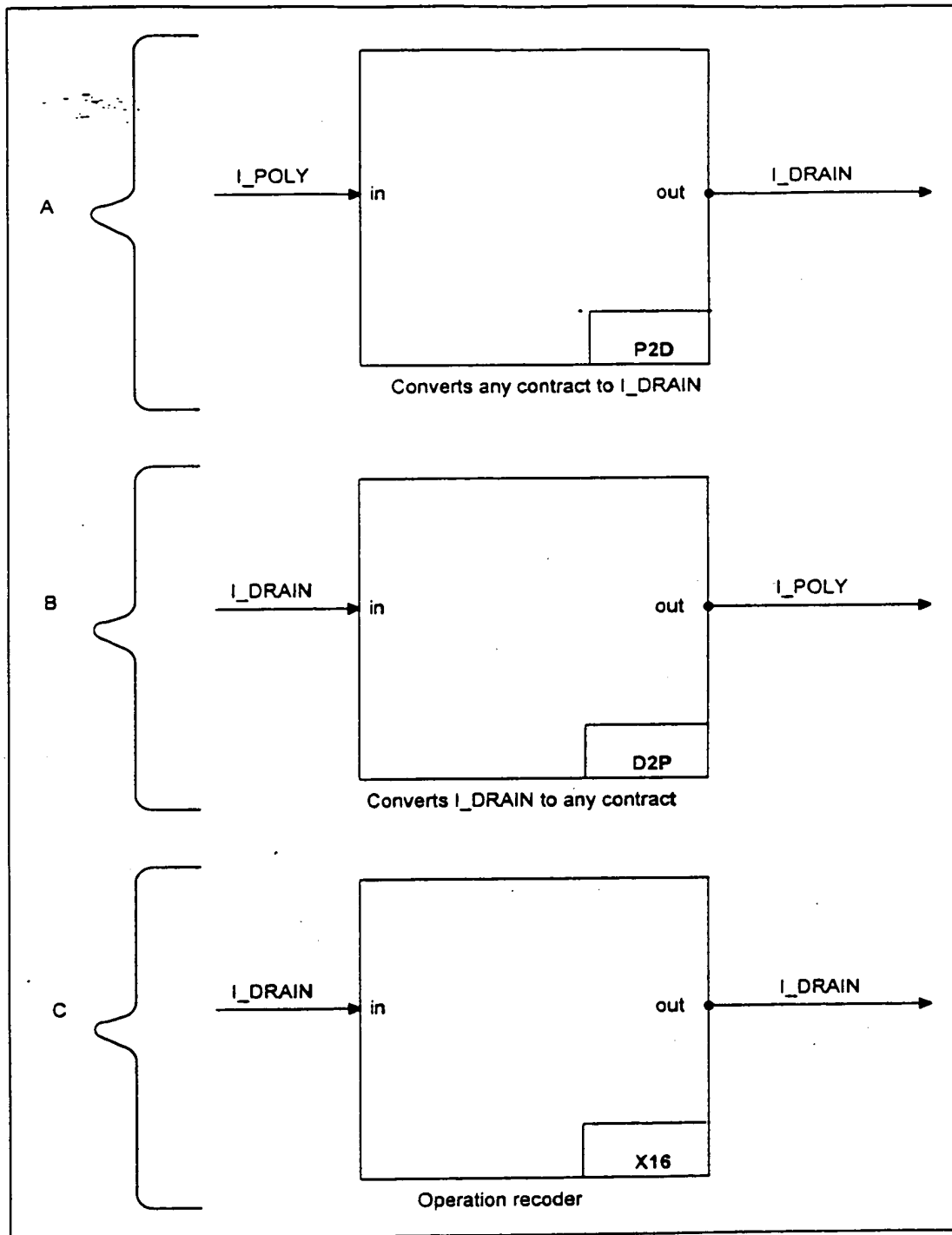


FIG. 30

